Freescale Semiconductor
Application Note

Document Number: AN3735
Rev. 1, 10/2008

MSC8122: Avoiding Arbitration Deadlock During Instruction Fetch

The MSC8122 DSP is a 4-core device based on the StarCore® SC140 DSP architecture, and is intended for router, multimedia, and gateway applications. The device bus architecture manages and arbitrates accesses between the 4 cores, the DMA controller, internal memory, and external memory.

The purpose of this document is to describe the possibility of deadlock due to bus starvation in the MSC8122 during code fetching, and provides guidelines to avoid deadlock and core starvation using code tuning and external memory timing.

NOTE

Although this application note is directed toward the MSC8122 DSP, the information can also apply to the MSC8112, MSC8113, and MSC8126 DSPs.

Contents

1. MSC8122 Device Overview (Bus Architecture) 2
2. MSC8122 SQBus Core Arbitration Scheme 2
3. Deadlock Cause During Instruction Fetch 3
4. How to Avoid Prefetch Related Deadlock 5
4.1. Using Code/Application Tuning 5
4.2. Asynchronous Prefetch Enabling 5
4.3. Avoiding the SQBus 5
5. Cautions About Altering Memory Access Timing . . . 5
A. Example Code for Enabling Cache, Write Buffer, and Prefetch for the MSC8122 7

freescale™ semiconductor

1 MSC8122 Device Overview (Bus Architecture)

The basic block diagram of the MSC8122 is shown in Figure 1

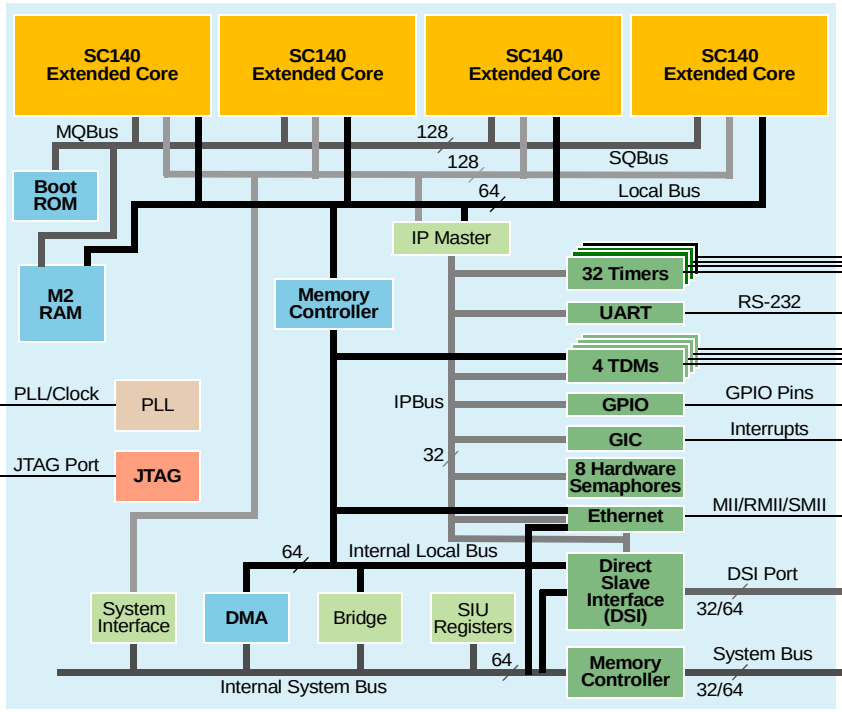

Figure 1. MSC8122 Block Diagram

This document focuses on MSC8122 memory accesses; specifically code fetching. At the device level, the four DSP cores in the MSC8122 access M2 memory via the MQBus and the Local Bus. Accessing external memory (off-chip SDRAM for example), requires using the SQBus. This application note focuses specifically on the SQBus and the effects of multicore access through the SQBus to external SDRAM through the System Interface Unit (SIU).

2 MSC8122 SQBus Core Arbitration Scheme

The SQBus uses 2 types of arbitration: round-robin arbitration and priority based arbitration.

- The round robin arbitration model guarantees access to each core that accesses the bus (at the same priority).
- Priority based arbitration provides higher access priority to certain accesses in the following order (from highest to lowest):
  - — High Priority:
    - – Non-prefetch data or instruction read accesses
    - – Immediate data write accesses
  - — Middle Priority—Data write access
  - — Low Priority—Instruction pre-fetch access

To prevent deadlocks (core freeze due to inadequate priority to access the bus) the SQBus arbiter and logic upgrades the priority of a core *middle priority request* when the core is frozen and loses access. If a core is frozen during a *low priority pre-fetch access*, the SQBus arbiter increases access priority *if the access is at a new cache line*.

Low priority pre-fetch requests however, are **not** upgraded to high priority during mid-cache line. The intent of this note is to clarify how the programmer should take care to avoid deadlock due to the case of a single core being unable to gain access to the SQBus during a mid-cache line pre-fetch while other cores have a higher priority on the SQBus.

3 Deadlock Cause During Instruction Fetch

The SQBus arbiter samples the priority of bus requests and gives access to the bus at a constant rate (1/8th the bus clock frequency). This section describes which scenarios can lead to deadlock for a core trying to perform a pre-fetch on the SQBus. To cause a deadlock, the following conditions must occur simultaneously:

- One core requests access to external memory via SQBus with low priority.
- Any other core or cores are accessing the SQBus with mid and high priority at every interval where the SQBus arbiter samples bus requests (see Figure 2).

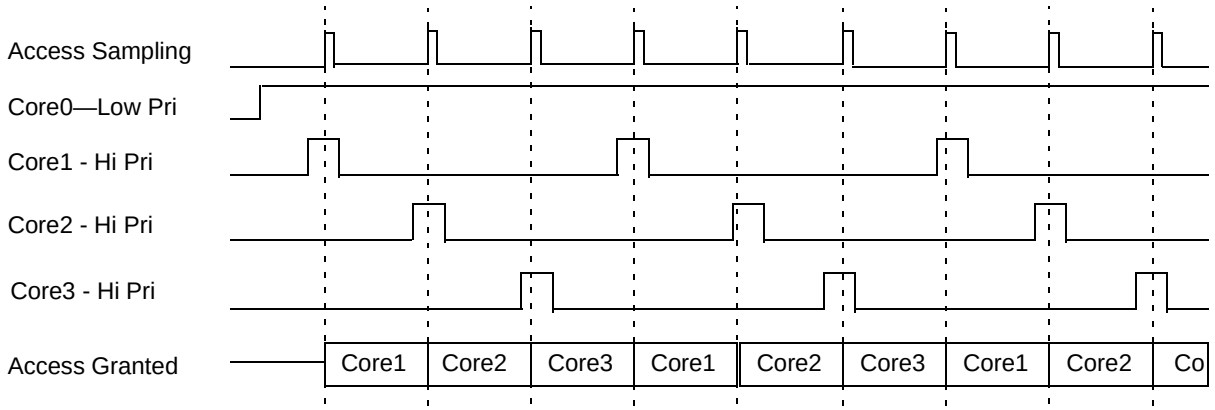

Figure 2. Cores Requesting Access: Core 0 Frozen (Never Granted Access)

Example 1 demonstrates the situation in which deadlock on one core may occur.

Example 1. Deadlock Scenario

The system runs the following setup: core 0 acts as a master core and initializes the device while the other 3 cores wait in a polling loop for the master core to finish.

In this code, the master core runs code from SDRAM (via the SQBus) using cache and prefetch. It is supposed to run a few NOPs, and then set a shared variable. The slave cores are running code directly from SDRAM (via SQBus, no cache), and poll a shared variable waiting for the master core to signal that the slave cores can begin executing.

NOTE

The problem here is that the slave cores are using the full bandwidth of the SQBus with constant high priority instruction access requests. The master core can gain access using prefetch (low priority) at the start of the cache line because the bus logic raises the master core priority after 1 freeze. But, as soon as a mid-cache line (low priority) prefetch begins, the core priority is not raised and the master core effectively freezes because when it starts the next prefetch, due to its low priority, the bus never grants access (see Figure 2).

Example 2. Deadlock Scenario Code

Master Core:

```
        org p:0
        jmp $20000000 ;Jumps to Code in SDRAM (Accessed via SQBus)
        org p:$20000000
    start:
    init: type func
        ;the following segment sets a shared variable in M2 to be used as a flag
        ;signals other cores that initialization is completed by Core 0
        move.l #$01010400,r5
        nop
        clr d0
        move.l d0,(r5)
        ;set breakpoint here and start slave cores before restarting master core
        bsr enable_icache
        bsr enable_wbuffer
        bsr enable_prefetch
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        move #1,(r5)
        debug ; only for Run/Stop at the end of the program debugging purpose
        nop
```

Slave Cores

```
         org p:0
        jmp $20F1E7FC
        org p:$20F1E7FC
        move.l #$01010400,r5
        nop
    test_loop:
        move.l (r5),d0
        tsteq d0
        nop
        bt test_loop
        nop
```

NOTE*

Functions/Macros such as “enable_icache” are defined in Appendix A, Example Code for Enabling Cache, Write Buffer, and Prefetch for the MSC8122.

4 How to Avoid Prefetch Related Deadlock

The following sections describe ways to avoid prefetch related deadlock situations.

4.1 Using Code/Application Tuning

The most basic way to ensure that priority related deadlock does not occur is to enable prefetch at the start of the application without intercore interdependencies. What this means is, the programmer needs to ensure that after one core has prefetch enabled, the other cores do not execute any looping code that depends on the first core before their respective prefetch units are enabled.

When there is no intercore dependencies before the prefetch instruction, an OS Barrier, and so forth, is not even necessary to ensure prefetch is enabled for all cores simultaneously. This is because the core(s) have prefetch enabled effectively hit a “priority barrier” by switching to low priority and getting blocked until the high priority cores reach their respective prefetch enable instructions.

4.2 Asynchronous Prefetch Enabling

If the programmer wants to enable prefetch for just one core (for some period of time) and use this core to initialize the device while other cores are polling without prefetch, deadlock can be avoided by running initialization code from non-SQBus memory.

4.3 Avoiding the SQBus

The obvious way to prevent deadlock on the SQBus is to not use it (that is, run prefetch-enabled code from M2 Memory and use the MQ Bus). You can do that using one of the following methods:

- In *assembly*: this is accomplished by allocating the initialization section of code to M2 or M1 using the `.org` directive.
- In a *C project*, the programmer can allocate the initialization sequence in a function to a section and locate this section in M1/M2 using the linker command file and pragmas, or the application file (`*.appli`). Refer to the StarCore C Compiler User Guide pdf file provided with CodeWarrior® IDE in the \help\PDF\ folder for more information on application files and pragmas.
- If M1 and M2 resources are limited, overlaying code is another approach. Overlay is the coding technique of copying sections of code from one memory location to another and then running that code.

5 Cautions About Altering Memory Access Timing

Testing showed that some level of deadlock can be avoided by altering timing settings to external memory, but this method has shown to cause bus overheating on ADS and custom boards, and as such should be strictly avoided. The theory behind this method is that the deadlock condition only occurs when there is overlap between accesses of different cores to external memory via the SQBus (as illustrated in Figure 2).

In a test using the MSC8122 ADS board, the overlap was eliminated by reducing delay parameters in SDRAM timing, as shown in Example 3.

Example 3. Adjusting SDRAM Timing

On the MSC8122ADS, SDRAM timing is configured in the `8122ADS_DSI32_Slave_Init.cfg` file generated by the CodeWarrior default stationary.

When running cores at 500 MHz (bus at 166MHz), and using the DSI configuration file set for 166 MHz SDRAM, asynchronous initialization using 1 core with prefetch can lead to deadlock.

In the test, deadlock was removed by decreasing the SDRAM delays to match the DSI configuration file 133 MHz SDRAM initialization.

However, AS A RESULT of the reduced delay in the SDRAM controller (required to eliminate access overlap), bus overheating occurred.

CAUTION

Adjusting the SDRAM timing to incorrect values for a certain bus frequency can result in bus overheating and potential damage to system boards. Do not use this method to attempt to resolve prefetch-related deadlock.

Appendix A Example Code for Enabling Cache, Write Buffer, and Prefetch for the MSC8122

```
enable_icache:
    move.l #pICCR,r1
    move.l #$0000f001,r0
    nop
    nop
    nop
    move.w r0,(r1)
    nop
    nop
    move.l #pICCMR,r1
    move.l #$00000001,r0
    nop
    nop
    nop
    move.w r0,(r1)
    nop
    nop
enable_wbuffer:
    move.l #WBCR,r1
    move.l #$000003FF,r0
    nop
    nop
    move.w r0,(r1)
    nop
    nop

enable_prefetch:
    move.l #pIFUR,r1
    move.l #$00000000,r0
    nop
    nop
    move.w r0,(r1)
    nop
    nop
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

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

Document Number: AN3735
Rev. 1
10/2008

freescale™
semiconductor