Freescale Semiconductor
Application Note

Document Number: AN3544
Rev. 0, 12/2007

PowerQUICC™ Data Cache Coherency

by *Gary Segal and David Smith*
Networking and Multimedia Group
Freescale Semiconductor, Inc.

Most processor based communication designs have a central processor core as well as other master devices, possibly within the same chip. The master devices typically provide a “low touch” method for the actual data transfer into or out of the communication system (DMA controller, MAC, etc.). However, if a system has multiple master devices, the designer must consider how cache coherency is maintained.

This application note provides an overview and specific strategies for cache coherency and potential performance impacts in the Freescale PowerQUICC II Pro and PowerQUICC III families. It focuses on data coherency and potential system issues, as well as some concerns associated with having multiple DMA devices. It also discusses the specific steps and interdependencies required to implement hardware enforced cache coherency.

Contents

freescale™ semiconductor

1 Cache Modes

Every cache implementation varies, but there are common modes and structures used by all designs.

A cache is a very fast memory located directly on the core processor's bus that is designed to provide data or instructions to the processor in significantly less time than main memory. When the core requests data, the cache first checks to see if it has such data. If it does, the cache provides the data back to the core with minimal clock cycles, which is called a cache hit. If the data is not in the cache, it fetches the data from main memory, provides it to the core, and saves it in the cache, which is called a cache miss.

Entries in the cache are marked as either "invalid" or "valid," indicating whether or not they hold usable data. At reset, boot software must ensure the entire cache is marked as invalid. As the core requests data, the cache fills. When all entries are full, the cache uses a replacement algorithm to decide what data to remove when new requests are made. The type of replacement algorithm can have a small performance impact, but is beyond the scope of this document.

When fetching data to fill a cache miss, the cache logic fetches more than the requested data. The block of data fetched is referred to as a cache line, which may be 32 to 128 bytes, depending on the system. The primary reason for fetching more than the requested data is due to the high probability that subsequent accesses are to nearby address locations. This behavior, know as "locality of reference," is the underlying principle that drives the performance enhancements provided by caches, but is beyond the scope of this document.

Another reason is to optimize the memory bus. DRAM, which is most often used for main memory, allows subsequent access to the same row to occur with much less latency than the first access. By reading an entire row from DRAM, the cache is able reduce the average latency used to access the DRAM, improving system performance. Most modern DRAMs take four or eight accesses, or beats, to read or write an entire row. This is called a burst read or write. When a burst access is performed, the amount of data transferred is a function of the bus width and number of beats. For example, a 64-bit wide data bus with a four beat burst transfers 32 bytes.

To reduce the complexity of address comparisons, cache memories are organized into small blocks called lines. Cache data is managed one line at a time. When selecting the line size, cache designers take the memory burst size into account. The cache line is a multiple of the basic burst size used by the memory system. Some caches have a line size equal to the burst size, while others may issue two, four, or eight bursts per line. When debugging cache issues, it is important to know the line size because any problems related to memory offsets that equal the line size are a strong indicator of a cache coherency problem.

When the processor writes data, caches respond in one of two ways. The first and simpler approach is called "write-through." When a cache operates in write-through mode, any data written by the processor is immediately written to main memory using the size of the operand. If the data also hits in a line in the cache, the cache may respond by either invalidating the entire cache line or updating the cache memory. Data written by the core (but not necessarily by other devices) matches in cache and main memory.

The other approach is called "copy-back" (sometimes called write-back). When a write hits in the cache, the cache line is updated with the new data. However, main memory is not updated, which means the cache and main memory will mismatch by design. The cache line is marked as "dirty" to indicate this condition. Data is written only from the cache when a read or write miss occurs. The cache selects a line to replace with the newly read data, and if the selected line is marked as dirty, it is written to memory. Because the

entire line is written, a burst write is used. Both burst reads and burst writes improve DRAM bus utilization. Note this is called a “cast-out.”

Copy-back mode should be used for the highest performance because it minimizes the external bus traffic (to the core). By design, copy-back creates mismatches between main memory and cache, and the system must take this into account.

2 Cache Coherency

Figure 1 shows a block diagram of a typical system with a PowerQUICC device with PCI connections.

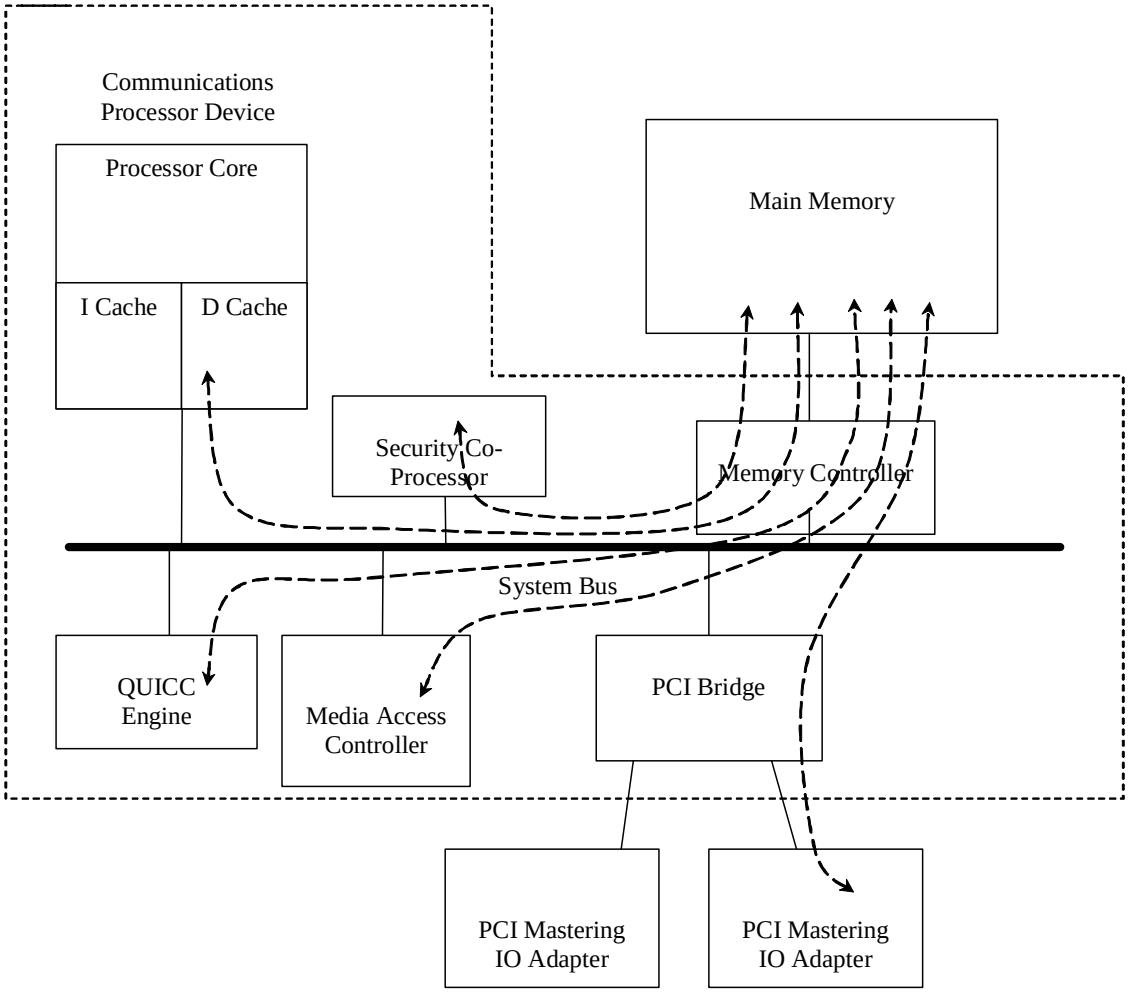

Figure 1. Typical System with Multiple Masters

To implement a system capable of efficient data transfer, networking designs typically comprise a system with multiple devices that can access main memory (referred to as direct memory access, or DMA capable devices). This ensures that the processor does not need to be involved in every data transfer within the system. Additionally, most modern processors also leverage caching schemes to reduce the access latency for information (data and instructions) required for execution. Depending on how the caches are managed,

the information physically stored in the cache may or may not match the information physically residing at the same address in main memory.

If the information in the cache matches the information in main memory when it is accessed by any master, it is said to be coherent. Typically instructions are only accessed by the core processor; the other DMA devices usually do not access instruction space other than to initially transfer the code into main memory.

An example of a potential cache coherency issue is the reception and subsequent processing of a block of data traffic. Typically this is handled by a DMA-capable media access controller device (MAC) that acts somewhat independently within the system. The MAC device may be within the communications processor (for example, Ethernet controller) or externally attached through a master capable bus interface, such as PCI or PCI-Express. The MAC device transfers the incoming data directly into main memory starting at a preprogrammed address location. After reception is complete, the core is tasked to access the data for processing. However, it is possible the core has previously accessed the memory location where the incoming data resides, and the information at that address resides (as far as the core is concerned) in cache. The core incorrectly fetches the data from cache (referred to as stale data), not the intended “fresh” data that resides in memory. Other scenarios exist where the more current or valid data resides in the data cache instead of main memory. Whenever the condition exists where the core and any of the DMA devices can access the same physical address, there is a potential for the memory to become incoherent. When a device processes invalid data, any number of issues can occur, often in a random and unpredictable way. Therefore, it is very important that data coherency be considered and a robust strategy is implemented to maintain coherency.

3 Avoiding Data Coherency Issues

This section describes how to best avoid various data coherency issues.

3.1 Non-Cached Data

The simplest way to avoid coherency issues is to not use data caching at all. However, this can cause a significant reduction in performance; some networking applications have been measured to run ten times faster with the data cache enabled versus disabled.

One strategy to avoid a coherency issue is to configure the system so that memory areas that can be accessed by multiple masters are not cacheable.

Another approach is defining specific memory areas as cache enabled just for the processor core with other shared memory areas with caching disabled. This requires appropriate configuration of the memory management unit (MMU) and a rigid definition of where the “core only” and “shared” memory spaces are located. Typically, data structures, such as the stack and OS heap, reside in the “core only” cacheable areas and data structures, just as the buffer descriptors and data buffers reside in non-cacheable memory. Coherency is maintained because the area that is accessible by multiple masters (core and DMA engines) is not cached and that memory area never resides in cache. Because the “core only” area is accessed only by a single master, there is no coherency concern for that area. This approach provides the cache performance boost for non-shared data, but does nothing for shared data. In systems that do significant processing on the shared data, not using the cache has a very negative impact on system performance.

3.2 Software Enforced Coherency

Using non-cached data structures is a very easy approach to guarantee coherency, but it significantly minimizes the performance advantages of a data cache. Managing coherency issues in software requires the software architect to have a clear understanding of how potential coherency issues can occur and how to implement additional cache management software to avoid them.

Some of the most common conditions that create potential coherency problems are as follows:

- A data buffer is created for transmission by the core and copy-back mode is used. In this case, dirty data in the cache may cause the DMA MAC device to see stale data in main memory.
- A data buffer is received (written by a DMA MAC) into a portion of main memory that is also cached. In this case the cache has stale data.
- Adjacent data buffers happen to share a cache line. In this case it is possible for software updating the “end” of one buffer to introduce a coherency issue with the next buffer.

With software enforced coherency, the device driver for the peripheral has to include specific instructions to manage the cache. Whenever the device driver prepares a buffer for transmission, it must flush the cache lines assigned to the buffer, which is done with specific cache management assembly instructions and must be done for each cache line the buffer occupies. This means an iterative loop that steps through the memory space used by the transmit buffer must be included in the device driver.

When the transmit buffer is flushed by the cache in response to cache management instructions, the core must be programmed to wait for the flush event to finish before continuing. This means the core wastes cycles while waiting for the flush. Because the flush is software initiated, the cache and bus logic are not free to use idle bus cycles to service the flush, further reducing bus utilization.

For receive, the system must guarantee the core does not read from the receive buffer space before the DMA MAC finishes reception. Otherwise, the memory is loaded into cache before the data received by the DMA MAC is written to memory.

To avoid the shared cache line problem, the software architect must carefully separate shared data structures to ensure cache lines are not inadvertently shared. Not only must cache line sharing be avoided (or carefully controlled) between transmit and receive data structures, it must also be prevented between general purpose memory and the peripheral memory. This means additional memory pools must be set up and managed, which can reduce overall memory system utilization.

Properly implemented software managed coherency requires both careful use of cache management instructions and memory space management by the programmer. In most cases, assembly language must be used to access the processor’s cache management instructions. The result is performance degradation, because while software managed coherency allows systems to maintain the performance advantages of caches, it introduces another level of complexity to the software. Over time, this complexity adds to development, debugging, and maintenance costs. If the system is upgraded to a processor with a different set of cache management instructions or a different cache line size, coherency software needs to be re-examined. If a new peripheral is added to the system, its coherency code and memory usage must be carefully integrated into the existing software architecture.

In summary, software managed coherency is a reasonable approach to obtain the performance benefits of caches at the expense of additional software development and maintenance costs. For systems without

hardware cache coherency, it is the preferred choice. However, if a system supports hardware coherency, it is best to avoid software managed coherency.

3.3 Hardware Enforced Coherency

An alternate method is to let the processor hardware enforce the system coherency. The processor core logic keeps track of the contents of its data cache and examines each transaction on the system bus. This is done to determine if there is a request to read from a location that resides in its cache or write to a location that it has in cache, and take the appropriate actions to ensure the coherency. Snooping is the mechanism the cache logic uses to examine transactions requiring coherency actions. There are two bus signals associated with snooping: the global signal (driven by the requester to indicate shared memory) and the retry signal (driven by the cache to indicate data to flush).

For Power Architecture™ processors, the data cache hardware implements a four state protocol (MESI) or three state protocol (MEI) to label each 32-byte cache line in any of the following states, each indicating the following:

- Modified (M)—The line is modified compared to system memory; the data is (currently) valid in the processor's cache but not in main memory, which is dirty.
- Exclusive (E)—The data in the cache line in the processor matches the system memory but does not exist within another processor's cache.
- Shared (S)—The data in the cache line is valid within the processor's cache, system memory, and at least one other processor's cache. This state only has meaning in systems with more than one processor core.
- Invalid (I)—The associated cache line does not have valid data within the cache.

Note that some processors do not implement the shared state, usually because there is no externally available coherent bus to enable connection of an additional processor.

When properly configured, the cache hardware snoops all the processor bus transactions and determines what actions are required to ensure coherency. The types of transactions that generally occur are as follows:

- Processor store—When the processor stores (or writes) data to main memory, the cache logic checks the address to determine if it resides in a cache enabled area and matches an address within the cache. If the address is cacheable and is already in cache, the cache line is updated. If the address is cacheable and not currently in the cache, the cache line is added using the cache replacement algorithm specified for the processor. If the cache is configured as write-through, the main memory is updated with the new data as well as the cache, and the line is marked exclusive (E). If the cache is configured as write-back, the data is not written to the main memory, and the line is marked modified (M).
- Processor load—When the processor loads (or reads) data, it checks for the data in cache. If the address resides in cache, the data loads directly from cache. If the data is not in cache, it is loaded from main memory to the cache and marked exclusive (E).
- DMA engine read—Assuming the proper snooping is configured when a DMA engine reads an address (described in Section 3.3.1, "Hardware Enforced Coherency for PowerQUICC II Pro and PowerQUICC III"), the cache logic checks to determine if the address resides in the data cache. If

the associated cache line is marked invalid (I), the transaction completes with no additional interaction because the data does not reside in cache. If the line is marked exclusive (E), implying the location is coherent, no additional bus interaction is required and the cache line is marked invalid (I) for burst cases (cache enabled transaction). If the line is marked modified (M) (most likely updated by the core with the cache configured as write-back), this indicates that the memory is currently not coherent with the cache. The cache snoop logic suspends the DMA read by forcing a retry back to the DMA device. The snoop logic pushes the cache data to main memory and marks it invalid (I), indicating the cache and memory are coherent. After this update, the transaction is allowed to complete and the DMA controller completes the read.

- DMA engine write—Assuming the proper snooping is configured, the cache logic snoops the transaction and checks to determine if the address resides within the cache. If the address is not within cache or within a line marked invalid (I), no action is taken. If the address is within the cache and the associated cache line is marked modified (M), the access is retried (similar to the DMA read case) and the cache line is cast out to main memory, guaranteeing coherency. The line is marked invalid (I) in the cache as the DMA write updates memory. If the address within the cache is marked exclusive (E), the line is marked as invalid (I) because the memory is no longer coherent with the cache.

By allowing the hardware to enforce the coherency, the software does not need to perform additional instructions to enforce the coherency. Most transactions complete without penalty. In the worst case, the cache logic forces a retry cycle on the bus when a DMA controller attempts to read or write a location that resides in cache and is marked modified (M). The overall transaction is extended, but if the system is using software coherency, a previous flush transaction is required before the DMA read; therefore, typically there is no appreciable system latency penalty with hardware coherency.

3.3.1 Hardware Enforced Coherency for PowerQUICC II Pro and PowerQUICC III

To enable hardware coherency in a PowerQUICC II Pro or PowerQUICC III system, the MMU and DMA controller(s) must be configured to enable the proper operation. The cache logic performs most of the action to enforce coherency; the DMA controller must be programmed to notify the logic when it is accessing memory locations that may be cached. The MMU is used to control the memory and defines the memory areas that are cached. The MMU mechanisms are either block address translation (BAT) or page table entries (PTE) through a translation lookaside buffer (TLB). In either case, there are 3 cache control bits:

- Caching inhibited (I)—If this bit is set, the area is not cached. There is no coherency issue, but performance is impacted significantly, so clear this bit.
- Write-through (W)—If this bit is set, the cache is configured as write-through; if cleared, the cache is configured as write-back. For most applications, write-back provides the best performance because the coherency hardware can selectively force the external bus traffic when necessary (MEI protocol).
- Memory coherence (M)—This bit is used to allow the core to broadcast when it is accessing memory located within another cache. Note that in single processor systems, this bit has no effect and is a “don’t care” because there is no other cache coherency logic to respond to the access.

For the PowerQUICC II Pro (MPC83xx) products, these bits are located in either bits 25–27 in the lower BAT register or bits 25–27 in the second word of the page table entry in the TLB. For the PowerQUICC III products, these are configured in bits 59–61 in the MAS2 register when the page table entry is defined into the TLB.

The DMA controllers must be configured to advertise accesses to potentially cached memory. This is done by enabling snooping by ensuring the global signal is driven on the processor bus (which is internal to the part for PowerQUICC II Pro and PowerQUICC III). Refer to the section describing the DMA controller(s) in the applicable reference manual for information.

Examples of these control bits are as follows:

- For “generic” DMA controller—DMA snoop enable (DMSEN) bit in the DMAMR*x* register
- For PCI controller—Read transaction type (RTT) and write transaction type (WTT) in the PIWAR*x* register
- For the Security Engine (SEC)—Global inhibit (GI) in the MCR register. Note that there is also a concept of snooping associated with the SEC that has to do with how the data is processed, this is not related to system cache coherency; the GI bit in the MCR is the only bit that controls the cache coherency.
- For the Triple Speed Ethernet Controller (TSEC)—Transmit data snoop enable (TDSEN) and transmit buffer descriptor snoop enable (TBDSEN) in the DMACTRL register, and receive data snoop enable (RDSEN) and receive buffer descriptor snoop enable (RBDSEN) in the ATTR register
- For the USB controller—SNOOP1 and SNOOP2 registers
- For QE peripherals—The global (GBL) bits in the Rx and Tx Bus Mode registers for the associated peripheral. The bus mode register is part of the TSTATE and RSTATE registers located in the peripheral’s parameter RAM.

4 Conclusion

When designing a processor based communication system, an effective cache coherency strategy must be implemented. Coherency issues can be avoided by defining areas accessed by multiple masters as non-cacheable, but this choice yields the lowest performance. Software may be written to enforce the coherency, but may have performance impacts due to unnecessary cast-out transactions. Writing software to enforce coherency tends to be cumbersome to implement and maintain, which can potentially introduce errors that are difficult to debug from a system level. In most cases, leveraging hardware enforced coherency is preferred, because the configuration is straightforward, the best performance benefit is realized, and special programming techniques are not required in the main stream software to maintain the coherency.

5 References

- *PowerPC™ Microprocessor Family: The Programming Environments for 32-Bit Microprocessors*
- *PowerPC™ e500 Core Family Reference Manual*
- AN3441: *Coherency and Synchronization Requirements for PowerQUICC™ III*
- EREF: *A Reference for Freescale Semiconductor Book E and the e500 Core*
- *e300 Power Architecture™ Core Family Reference Manual*

6 Revision History

Table 1 provides a revision history for this application note.

Table 1. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	12/2007	Initial release.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

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 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 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN3544
Rev. 0
12/2007