Achieving Cache Coherence in a MIPS32® Multicore Design

Document Number: MD00888
Revision 01.00
August 17, 2008

MIPS Technologies, Inc.
995 East Arques Avenue
Sunnyvale, CA 94085-4521

Introduction

Scaling processing performance beyond the frequency and power envelope of single core systems has led to the emergence of multi-core clusters. Data access management within such processing systems becomes essential to ensure behavioral consistency. One solution to provide access consistency is the application of a memory coherence model such as MESI or MOESI within the L1 data cache hierarchy. For the MIPS Technologies MIPS32® 1004K™ Coherent Processing System (CPS), we -applied Open Core Protocol (OCP) point-to-point connectivity to establish snoop-based coherence throughout the cluster. Following are principles of this communication model.

Deriving a Message-based Memory Coherence Model

Historically, memory coherence in multiprocessor systems was often achieved through bus 'snooping,' where each core was connected to a common multi-tier bus and was able to snoop on memory access traffic of processor peers to regulate the coherence status of individual cache lines. For that, each core maintained the coherence status of L1 cache lines locally, and posted status changes to peers via the common bus.

The increasing size and complexity of SoCs led to restructuring of the multi-tier bus philosophy in favor of localized point-to-point connections with centralized traffic routing. This allowed dramatic speedup and power improvements on now localized bus segments due to reduced load and segment length. Also, bus contention problems eased, and throughput increased for the localized data exchange. In response to this system architectural trend, the Open Core Protocol (OCP) standard emerged to consolidate this design philosophy. Further, emergence of IP provider business models catalyzed the standardization of IP interconnect and design methodology to facilitate design reuse centered on an open standard.

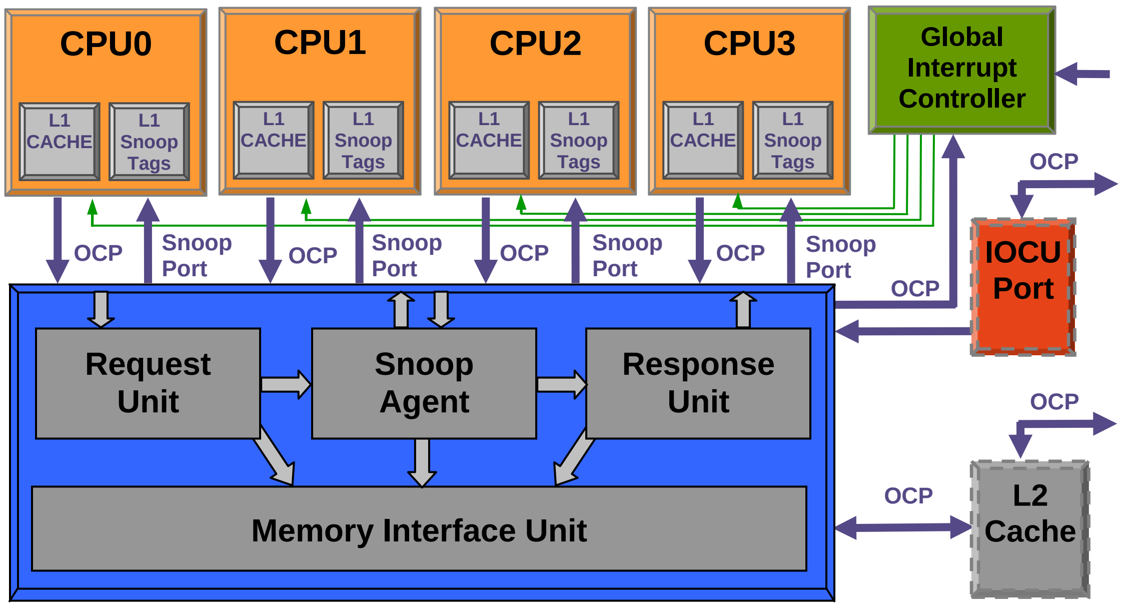


Figure 1: Coherent Processing System

However, localized bus transactions, as conducted through OCP interconnect segments, decouple processors throughout a multi-core cluster. Coherence schemes cannot be directly based on bus snooping and reliance on bus arbitration to ensure access ordering. Different methods of communication are needed to ensure data access consistency. Additional challenges arise in the ordering of competing L1 line data requests. One way to addresses these challenges is to add coherence message communication to each processing element as depicted in Figure 1. These messages provide the means of snoop type cache coherence.

Coherence messages embody a new type of command within the OCP protocol. Members of the processor system send coherence messages toward a centralized coherence manager that provides access ordering (serialization) and message routing to provide snoop-type access to peer members. These peers will respond with their individual L1 line status and post a message response. Depending on responses, the coherence manager initiates data movement for coherent data between cores, and funnels access toward higher-level memory hierarchies such as L2 and L3 caches. I/O coherence units also provide a means to phase-in/out data toward/from the coherent address space, and are part of coherent message exchange.

In addition to new message-type commands within the OCP protocol, individual processors are required to respond to coherent status requests, and are therefore not solely initiators (masters) of bus transactions. The coherent processing system might address this requirement by providing an OCP slave port to receive and respond to messages initiated by the coherence manager. Coherent requests by a processor will utilize the OCP master port. Within the processing cluster, coherence message exchanges between cores and the coherence manager are dubbed ‘interventions.’ OCP slave ports of processors receiving interventions are therefore ‘intervention ports.’

As depicted in Figure 1, each individual processor of the 1004K system is based on our multi-threaded processor architecture, providing two independent threads and processing context within the envelope of a single-scalar, 9-stage pipeline. Level 1 data cache tag arrays are duplicated to be accessible simultaneously for CPU operation and intervention lookup. MESI style cache line coherency is supported.

The coherence manager of the processing system receives and serializes incoming messages through its request unit – OCP slave ports, driven by each CPU and I/O-coherence units. Serialized messages are routed depending on their address space and context either to higher-level cache hierarchies using the ‘Memory Interface Unit,’ or toward processor peers and I/O-coherence units using the ‘Snoop Agent.’ The snoop agent initiates OCP master transactions (interventions) to look up the coherent L1 cache line status for each processor. Interventions returned to the initiator of a message, called self-interventions, allow the initiator to provide access ordering. Responses to coherent messages initiated by CPUs as well as data responses are formulated within the ‘Response Unit’ and routed to individual CPUs.

Coherent OCP Commands

OCP commands used within the 1004K CPS can be classified into three categories.

First are the **Coherent Messages** that maintain a MESI-style cache line status. These are a result of CPU load/store operations and can initiate data movement between CPUs and/or the memory subsystem. All peer CPUs of the CPS will receive coherent messages posted by an initiator, and respond according to their cache line coherent state. The coherence manager will initiate data movement as required.

Coherent Cache Manipulation Commands are utilized for cache line maintenance within the coherent address space. I/O traffic will bring new coherent lines into the domain or remove coherent context from cache lines. Further, memory hierarchy synchronization operations are performed.

The third category is **Non-Coherent Commands**, which perform OCP main port transactions on memory regions outside the coherent address space. These represent OCP read and write commands.

Coherent Messages

The coherent processing system may implement four coherent messages that are caused by L1 cache line status changes due to CPU load/store activity. The initiating CPU sends this message as an OCP master port command. Peer CPUs of the system receive interventions based on this line status change and will respond with their local cache line status.

The first message type is the **CohReadOwn**, denoting a cache miss that occurred through an attempt to modify a cache line. Peer cores encountering this line in status 'Modified' will force a write-back into the memory subsystem and perform a local invalidate. As an optimization, locally encountered line data will be forwarded to the requester CPU to reduce access latency. The requester CPU will install this line as 'Exclusive' and perform the line modifying instruction. Then the cache line status will change to 'Modified.' While waiting for line refill, the requester CPU will continue execution of another thread.

The **CohReadShared** message indicates that a cache miss occurred through a line read operation. No line modification is intended. Peer cores encountering this line in status 'Modified' will force a write-back into the memory subsystem. Hitting peer lines will migrate to 'Shared' status. Hit data is forwarded to the requester core and installed in state 'Shared.' Then the line read operation is performed. While waiting for line refill, the requester CPU will continue execution of another thread.

CohUpgrade indicates that a line modifying instruction encountered a cache hit on a 'Shared' line. Peer cores will be notified to invalidate hitting lines. The 'Shared' line is then upgraded to 'Modified' after the modifying instruction is executed.

Finally, the **CohWriteBack** message signifies eviction of a coherent cache line. The coherence manager will initiate data movement through the intervention port and forward data to the memory subsystem. The evicted cache line is then replaced by a new – possibly coherent – address. In this case, a CohReadOwn or CohReadShared has caused the eviction.

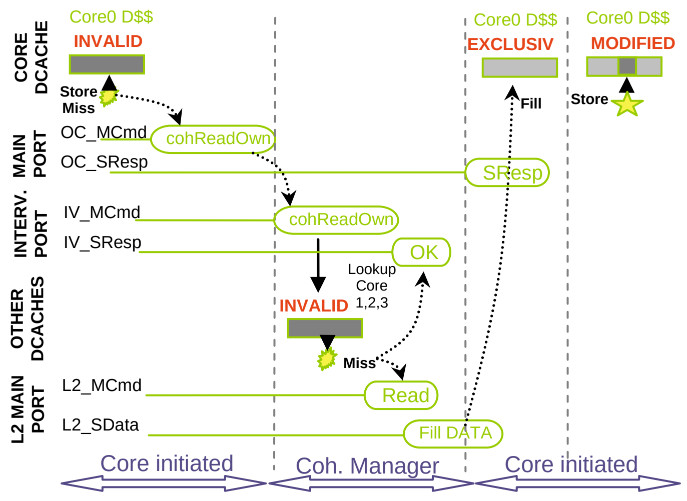
Coherent Cache Manipulation Commands

In response to cache manipulations, coherence messages are initiated and sent to peers.

- **CohCopyBack** – write back a coherent cache line to the memory subsystem. Cache line hits in state 'Modified' will be written back. Line status migrates to 'Shared.' CopyBack data movement will be initiated by the coherence manager using the intervention port.
- **CohInvalidate** – purge a coherent cache line without writing back its contents to the memory subsystem. This command is always data-less and is posted to each peer of the CPS. Invalidate type cache operations cause a CohInvalidate message.
- **CohWriteInvalidate** – an I/O coherence unit injects a new cache line into the coherent domain. Existing peer line data will be invalidated throughout the CPS.
- **CohReadInvalidate** – an I/O coherence unit notifies the system about a cache line leaving the coherent domain. Existing peer line data will be invalidated throughout the CPS.
- **CohCompletionSync** – data-less command to maintain ordering. Local buffers of CPS peers are flushed towards the memory subsystem. The CPU-SYNC instruction causes the CohCompletion-Sync for CPUs attending the coherent domain. SYNC command arguments (sync types) help control the depth of flush operations throughout memory hierarchies. The coherent processing system reserves certain argument encodings to support low overhead access ordering.

Non-Coherent Commands

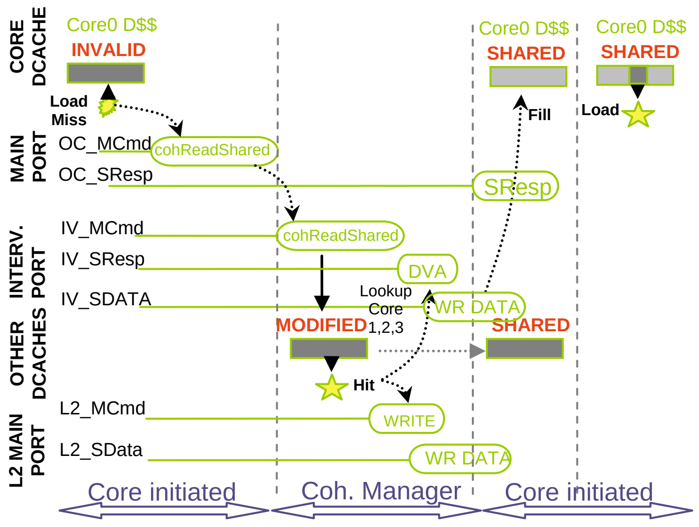
Traditional OCP commands such as 'Read' and 'Write' are supported throughout the coherent processing system to handle data access for non-coherent memory access. The **Read** command is issued when a miss within a cached, non-coherent address or an un-cached access causes a read operation from the memory subsystem. Response data – if cacheable – will be installed as non-coherent, whereas un-cached data are consumed directly. Fetch as well as load/store activity causes Read transactions. The **Write** command is issued when cached, non-coherent eviction data, or un-cached address range stores will be written back to the memory subsystem. The OCP main port of a core performs the command and data phases of the transaction.



Example – CohReadOwn

CPU 0 encounters a store miss and initiates a cohReadOwn (intent to modify) message toward the coherence manager. The coherence manager sends interventions toward all cores. None of the peers have this cache line available, and an OCP read request is directed toward the L2 cache. Returning data will be installed with the coherence attribute 'Exclusive' at the requester core. After the store operation completes, the cache line status migrates to 'Modified.'

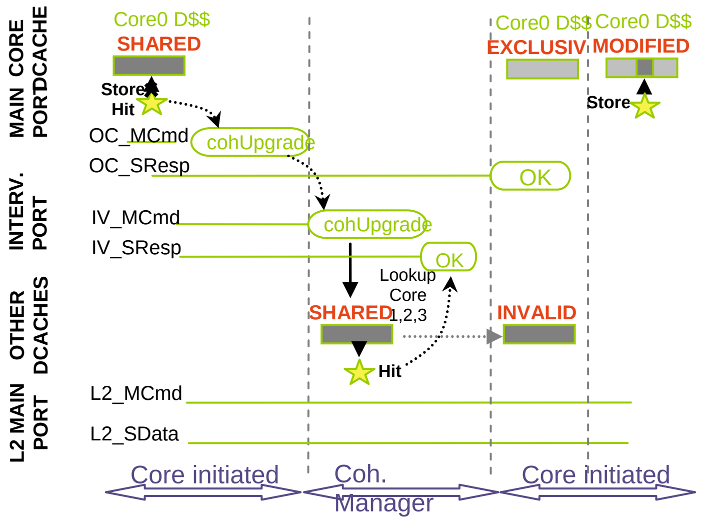
Figure 2: Coherent Read Own Messaging



Example – CohReadShared

CPU0 encounters a load miss on a coherent cache line and initiates a cohReadShared message (no intent to modify). The coherence manager sends interventions to all cores where core 1 responds with a hit – 'Modified.' The coherence manager now initiates a write-back of the modified line, and moves line data from the core 1 intervention port to the memory subsystem. The hitting Core 1 cache line migrates to 'Shared' status. Line data movement also forwards to core 0 where it is installed in the 'Shared' state.

Figure 3: Coherent Read Shared Messaging



Example – CohUpgrade

Core 0 encounters a store hit on a 'Shared' marked cache line. A cohUpgrade request is sent and the coherence manager initiates interventions to all cores. Core 1 responds with a hit 'Shared' and invalidates its line. Core 0 is permitted to upgrade its cache line to 'Exclusive.' After the store has completed, the cache line status migrates to 'Modified.' State 'Exclusive' is required (rather than 'Modified' immediately) since between intervention response and store execution, other cores could intervene and force a modified write-back—migrating to 'Shared' without awaiting the core 0 store.

Figure 4: Coherent Upgrade Messaging

Conclusion

The Open Core Protocol (OCP) interconnect lent itself well to support message-based coherence implementations. A centralized coherence manager serializes coherence messages emanating from an individual core and inquires about the coherence status of peer cores. Data forwarding between cores decreases access latency and reduces traffic to higher levels of memory hierarchy. Individual cores posses an OCP master port to initiate data access and an OCP slave port to receive inquiries from the coherence manager.