

9800707-3
IPos copy

- 1 -

**METHOD OF LAYERING CACHE AND ARCHITECTURAL
SPECIFIC FUNCTIONS TO FACILITATE DESIGN EXTENSION**

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates in general to cache controllers in data processing systems and in particular to cache controllers which layer cache and architectural specific functions. Still more particularly, the present invention relates to supporting straightforward extension of nonshared cache controller implementations to shared implementations, with simple resolution of associated flow rate issues.

2. Description of the Related Art:

Data processing systems which utilize a level two (L2) cache typically include a cache controller for managing transactions affecting the cache. Such cache controllers are conventionally implemented on a functional level, as depicted in Figure 7. For example, a cache controller 702 may include logic 704 for maintaining the cache directory, logic 706 for implementing a least recently used (LRU) replacement policy, logic for managing reload buffers 708, and logic for managing store-back buffers 710. In traditional implementations, the cache is generally very visible to these and other architectural functions typically required for cache controllers, with the result that cache controller designs are specific to a particular processors such as the PowerPC™, Alpha™, or the x86 family of processors.

The basic controller structure depicted in Figure 7 is employed in both nonshared and shared cache controller designs. Nonshared cache controllers manages transactions between a single upstream processor and a nonshared cache. Shared cache controllers, which manage transactions to a cache shared by two or more processors, are difficult to design but have been successfully implemented. Because of the extensive interlocks required, controller designs for a single upstream processor are virtually useless when designing a controller for a shared cache. The logic required for nonshared cache controllers is already complex for many processors and bus architectures because of the numerous special cases which must be accommodated. The logic complexity increases exponentially as additional upstream processors are added and additional interlock logic for coordination becomes necessary. In

particular, a substantial portion of the complexity in a shared cache design derives from the requirement--unique to shared cache designs--that the cache controller be capable of responding to a plurality of operations initiated by different processors in the same cycle. For a nonshared cache, only one operation may be initiated within a given cycle. Under the prior art approach, nonshared cache controller designs may not be easily reused in shared cache controllers, but require extensive and laborious design modifications and enhancements for successful transition to a shared controller design.

An additional problem with prior art designs relates to operation flow rate. Typically operations are initiated at a certain rate. In traditional implementations, operations initiated by an upstream processor or processors generally cannot be cancelled or ignored by the cache controller. The cache controller must respond to the operation, even if only to indicate that the operation should be retried later. Therefore, cache controllers are designed to respond to operations at the rate at which they are initiated by an upstream processor or processors. Thus, if a controller supports a single upstream processor which initiates operations at a given flow rate, cache controller 702 must respond to operations at that flow rate. For example, where a bus is defined to be capable of producing an operation once every three cycles, the cache controller is also designed to respond to operations at a rate of one every three cycles. Similarly, shared cache controllers are designed to respond to operations at a cumulative flow rate for upstream processors (that is, equal to the flow rate of all operations initiated by processors sharing the subject cache). Such operational speed requirements, imposed on necessarily complex logic, present a

substantial design obstacle.

It would be desirable, therefore, to implement a cache controller which permits straightforward extension of a nonshared controller design to a shared design. It would further be advantageous to provide a cache controller design permitting uncomplicated resolution of the operation flow rate issues associated with implementing a shared cache controller.

THIS SHEET HAS BEEN
REPLACED

SUMMARY OF THE INVENTION

It is therefore one object of the present invention to provide an improved cache controller for a data processing system.

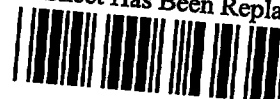
It is another object of the present invention to provide an improved cache controller having layered cache and architectural specific functions.

It is yet another object of the present invention to provide a cache controller design supporting straightforward extension of nonshared cache controller implementations to shared implementations, with simple resolution of associated flow rate issues.

The foregoing objects are achieved as is now described. Cache and architectural functions within a cache controller are layered and provided with generic interfaces, isolating controller logic from specific architectural complexities. Controller logic may thus be readily duplicated to extend a nonshared cache controller design to a shared cache controller design, with only straightforward modifications required. Throttling of processor-initiated operations handled by the same controller logic resolves operation flow rate issues with acceptable performance trade-offs.

The above as well as additional objects, features, and advantages of the present invention will become apparent in the following detailed written description.

This Sheet Has Been Replaced



BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts a data processing system implemented with a nonshared cache controller design in accordance with a preferred embodiment of the present invention;

Figure 2 is a block diagram of a nonshared cache controller design in accordance with a preferred embodiment of the present invention;

Figure 3 depicts a data processing system implemented with a shared cache controller design in accordance with a preferred embodiment of the present invention;

Figure 4 is a block diagram of a shared cache controller design in accordance with a preferred embodiment of the present invention;

Figure 5 depicts a timing diagram for a stalling mechanism which may be employed by a shared cache controller in accordance with a preferred embodiment of the present invention;

Figure 6 is a high level flowchart for a process of

managing operations in a shared cache controller in accordance with a preferred embodiment of the present invention; and

Figure 7 depicts a prior art cache controller design.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures, and in particular with reference to Figure 1, a data processing system implemented with a nonshared cache controller design in accordance with a preferred embodiment of the present invention is depicted. Data processing system 100 may include only a single processor or may be a symmetric multiprocessor (SMP) system including a plurality of processors. A single processor system is shown in the example depicted. Processor 102 may be a superscalar reduced instruction set computing (RISC) processor including separate level one instruction and data caches 104 and 106 within the processor. A PowerPC™ processor may be utilized for processor 102.

Processor 102 is connected to a level two (L2) cache 108, which is a nonshared cache. A second processor (not shown) may be added to the system depicted, provided a separate L2 cache, if any, is utilized by the second processor. L2 cache 108 is connected to system bus 110 for data processing system 100. Local memory 112 is also connected to system bus 110, as is I/O bus bridge 114. Other devices, such as memory-mapped graphics adapter 116, may also be connected to system bus 110. I/O bus bridge 114 is connected to I/O bus 118, which may be connected to a variety of other devices such as local area network (LAN) adapter 120 and hard disk drive 122.

Those of ordinary skill in the art will appreciate that the hardware depicted in Figure 1 may vary. For example, other peripheral devices, such as optical disk drive and the like also may be used in addition or in place of the hardware depicted. The depicted example is not meant to imply

architectural imitations with respect to the present invention. In particular, a data processing system need not be limited to a single processor as shown in the depicted example to benefit from the present invention. The present invention may be employed, for example, to improve the performance of a data processing system having two processors, each with a corresponding L2 cache.

Referring to Figure 2, a block diagram of a nonshared cache controller design in accordance with a preferred embodiment of the present invention is illustrated. Controller 202 is implemented within cache 108 depicted in Figure 1. Controller 202 includes a bus interface unit (BIU) 204 connected to an upper bus 206 for a processor or a higher level cache, and a bus interface unit 208 to a lower bus 210, which may be a system bus or a bus to another cache. Upper bus 206 and lower bus 210 may differ; upper bus 206 may be, for example, a 60X bus, while lower bus 210 may be a different bus.

Cache and architectural specific functions within controller 202 are layered. Thus, controller 202 includes cache controller 212 and architectural controller 214. Operations are distinguished as "cache" or "architectural" operations. Only cache operations are handled by cache controller 212, and only cache controller 212 performs operations on cache 216. Architectural operations are handled by architectural controller 214 and are seen by cache controller 212 as system-side operations.

A third unit, noncacheable controller 218, is also contained within cache controller 202. Noncacheable

controller 218 is actually a counterpart to cache controller 212 in that it also handles only cache operations. Whereas cache controller 212 handles cache operations directed at cache memory locations, noncacheable controller 218 handles cache operations directed at memory locations which do not map to cache 216. It is advantageous, for reasons known to those skilled in the art, to treat part of the system memory as noncacheable. Such memory may be utilized, for example, by memory mapped devices. While cache controller 212 operates on full cache blocks, noncacheable controller 218 operates on smaller memory segments, typically less than 8-16 bytes. Moreover, noncacheable controller 218 does not store data, while cache controller 212 retains copies of data handled within cache 216.

Cache operations are typically those operations which read or write values to memory locations, and therefore may change or retrieve the value of data in a memory location. Remaining operations are defined as architectural operations. Unlike cache operations, architectural operations generally do not change the value of data in a memory location. An architectural operation may move the data to a different location within the cache hierarchy, change the status of data in a particular memory location, or perform other such functions. However, architectural operations generally do not directly alter the value of data within a memory location.

Cache operations, supported by cache controller 212, comprise the largest majority of operations affecting the system cache. Within the complete set of operations supported by a given processor, cache operations may not derive from the portion of the instruction set which is most frequently

executed and/or consume the largest majority of processor cycles. However, disregarding instructions directed to other functional units within the processor, such as the floating point, fixed point, or branch units, cache operations are, collectively, executed most often and utilize the largest measure of time.

The remaining operations affecting a system cache--those employed for cache management, operating system management, page management, and synchronization, etc.--are layered out and supported by architectural controller 214. Virtually all processor architectures support such operations, which are utilized in real time operation much less frequently than cache operations. Additionally, individual operations among the architectural operations are generally implemented, if at all, in substantially divergent manners for different processors of interest. Processor-side architectural operations pass through architectural controller 214 to system bus 210 and affects cache controller 212 as apparent system-side architectural operations.

Different designs may vary the set of operations supported by the cache controller and, by default, the remaining operations layered for support by the architectural controller. However, increasing the number of operations supported by the cache controller increases the complexity of logic required. Additionally, if instructions selected for support by the cache controller are not supported by all processors of interest, the cache controller design loses its direct transferability to new controller designs.

While certain operations pass down only one path within

controller 202--that is, through architectural controller 214 or cache controller 212--other operations are split and pass down both paths. Cache controller 212 employs a pass-through design, in which operations initiated at interface 220 generate a response at interface 222 while operations initiated at interface 222 produce a responsive action at interface 220.

Because cache and architectural operations are layered within controller 202, bus transactions and protocols may also be layered. That is, generic interfaces may be defined for cache controller 212, architectural controller 214, and noncacheable controller 218. Thus, interfaces 220-230 comprise generic protocol interfaces to bus interface units 204 and 208 which are, to the extent possible, not architecturally specific. This decouples the design for cache controller 212 from the specific protocols of bus 206 and bus 210, allowing the design for cache controller 212 to be reused. Bus interface units 204 and 208 are responsible for managing transactions and protocols to bus 206 and system bus 210, translating the specific bus transactions into the protocol for generic interfaces 220-230. By employing generic interfaces for interfaces 220-230, the designs for controllers 212, 214, and 218 are isolated from specific bus architectures and may be readily duplicated.

In contrast to traditional cache controllers, cache controller 212 may thus be implemented in a manner independent of the two buses 206 and 210, responding only to cache operations. Although such cache operations are initiated by transactions on either bus 206 or bus 210, only certain bus transactions will prompt a response within cache controller

212. In a preferred embodiment, cache controller 212 only responds to instruction fetch operations (IFETCH), LOAD operations, and WRITE operations on bus 206, and to READ operations, WRITE operations, and traditional SNOOPS on bus 210. This results in substantially simplified design requirements for cache controller 212. This is accomplished by avoiding the usual practice of overlaying the highly irregular (semantically and temporally) architectural operations and cache operations. The burden of responding to the architectural operations is removed from the design of cache controller 212 and placed in architectural controller 214.

The cache operations handled by cache controller 212 are supported by every commercial processor of interest in substantially the same form. Only minor differences in specific implementation, from which cache controller 212 in the present invention is decoupled by generic interfaces 220 and 222, distinguish comparable instructions for different processors of interest.

By layering selected cache and architectural functions, and implementing generic interfaces to bus interface units 204 and 208, a large portion of the overall design of controller 202 may be directly transferred to new implementations. The cache controller logic may be reused without modification for cache operations. New sleeves of logic for the bus interface units may be easily implemented for handling new bus protocols and converting the generic protocol interfaces 220-230 of cache, architectural, and noncacheable controllers 212, 214, and 218 to interfaces for bus 206 and bus 210. The most significant effort for implementing a design supporting a

different processor is required by the architectural controller. Individual design of the logic supporting the architectural operations is required in any case since processor architectures vary dramatically. Overall, however, a significant savings in design effort for different processors may be achieved since only the semantics of operations handled by architectural controller 214 will change.

By layering cache and architectural functions, limiting cache controller 212 to responding to a few fundamental operations, the cache controller logic is greatly streamlined and simplified. In addition, the architectural controller logic is also simplified since, by separating the two classes of operations, issues of interrelationships between operations in different classes are eliminated. The cache and architectural controllers may be designed as individual units.

With reference now to Figure 3, a data processing system implemented with a shared cache controller design in accordance with a preferred embodiment of the present invention is depicted. Data processing system 300 includes a plurality of processors 302 and 304. Processor 302 in the depicted example includes separate internal level one (L1) instruction and data caches 306 and 308, respectively; similarly processor 304 includes separate L1 instruction and data caches 310 and 312, respectively.

Processors 302 and 304 are both connected to a single level two (L2) cache 314, which is a shared cache. L2 cache 314 is connected to system bus 316 for data processing system 300. Local memory 318 is also connected to system bus 316, as

is I/O bus bridge 320 and memory-mapped graphics adapter 322. I/O bus bridge 320 is connected to I/O bus 324, which may be connected to a variety of other devices such as local area network (LAN) adapter 326 and hard disk drive 328.

As with Figure 1, those skilled in the art will appreciate that the hardware depicted in Figure 3 may vary, and that the depicted example is not intended to imply architectural imitations with respect to the present invention.

Referring to Figure 4, a block diagram of a shared cache controller design in accordance with a preferred embodiment of the present invention. Controller 402 is formed from the same components comprising controller 202 depicted in Figure 2, with architectural controller 214, noncacheable controller 218, and processor-side bus interface unit 204 being duplicated. Thus, controller 402 includes two architectural controllers 404 and 406, two noncacheable controllers 408 and 410, and two processor-side bus interface units 412 and 414. A single cache controller 416 has the exclusive ability to perform operations on cache 418.

A single system-side bus interface unit 420, connected to system bus 422, is connected to each of controllers 404-416 via generic interfaces 424-432. One processor-side bus interface unit 412, connected to a bus 434 to one processor (not shown), is connected to one architectural controller 404, one noncacheable controller 408, and cache controller 416 via generic interfaces 436-440, respectively. The other processor-side bus interface unit 414, connected to a second bus 442 to a second processor (not shown), is connected to the other architectural controller 406, the other noncacheable

controller 410, and cache controller 416 via generic interfaces 444-448. Thus, cache controller 416 includes duplicate processor-side generic interfaces 440 and 444, each to a different processor-side bus interface unit 412 or 414.

Cache controller 416 requires more logic than cache controller 212 depicted in Figure 2. However, the cache controller design is layered into functional units corresponding to specific operations to be handled. Each functional unit is substantially autonomous and handles one type of operation, such as a READ operation. Nominal interlocking between functional units is provided to prevent functional units from overlapping. Within a group of equivalent functional units for one type of operation, the interlocking logic determines which functional unit will handle a specific operation, preventing others from also performing the same operation. Thus, cache controller 416 includes the same functional units as were utilized in the design of cache controller 212 depicted in Figure 2, but simply adds additional instances of given functional units.

Bus interface unit 420 requires more ports than the corresponding bus interface unit 208 depicted in Figure 2. Again, the BIU design is layered into multiple, substantially autonomous units which may be easily replicated. Thus, additional ports may be easily implemented within bus interface unit 420.

In extending the nonshared controller design of Figure 2 to the shared controller design of Figure 4, a correspondence between ports in bus interface unit 420 and generic interfaces 424-432 is required. That is, while bus interface unit 408

depicted in Figure 2 was only required to provide ports supporting three groups of interfaces 222, 226, and 230, bus interface unit 420 must provide ports supporting five groups of generic interfaces 424-432. This may be accomplished by either reducing the number of instances within the groups of generic interfaces to equal the number of ports available, or increasing the number of ports to equal the number of generic interfaces implemented. The option selected is a matter of design choice since layering of both the bus interface unit 420 and controllers 404, 406, 408, 410, and 416 makes both alternatives simple to execute.

Throttle unit 450 addresses the flow rate issues of extending a nonshared design to a shared design. Throttle unit 450 intercepts most of the address and control signals 452 and 454 forming buses 434 and 442, allowing the data portions 456 and 458 of those buses to pass directly to bus interface units 412 and 414. Throttle unit 450 stalls operations as necessary to prevent the overall flow rate of operations handled by cache controller 416 from exceeding the maximum supported flow rate. This allows the design of cache controller 212 to be reused in cache controller 416 with only relatively simple modifications.

Throttle unit 450 also serializes operations from the two processors. If two operations initiated by different processors temporally collide, such as when both are initiated in the same cycle, throttle unit 450 delays one operation while the other proceeds. Operations initiated by processors attached to buses 434 and 442 are thus serialized and delayed to the extent required by controller 402. Therefore, the fundamental design of cache controller 212 depicted in Figure

2 may be reused in cache controller 416 despite a probable increase in operation flow rate. Operations are stalled when necessary to reduce the flow rate to an acceptable level for cache controller 416.

Several well known techniques for deciding which operation to stall may be employed by throttle unit 450. For example, throttle unit 450 may simply select one of buses 434 and 442 to always lose when operations are initiated on both buses simultaneously. Alternatively, throttle unit 450 may employ some dynamic mechanism to vary which bus 434 or 442 loses for a particular collision, such as pseudo-random or round robin selection. These and other approaches are well known to those skilled in the art.

The present invention is limited by the requirement that the bus protocol for buses 434 and 442 tolerate being stalled. Not all bus protocols support such an option. However, most bus protocols of interest, including the 60X bus, support the ability to stall operations because frequently bus participants have varying performance levels.

Although depicted as implemented within a level two cache shared by two processors in Figure 4, the invention described is equally applicable to equivalent configurations, such as a level three cache shared by two L2 caches. The exemplary embodiment employed to describe the present invention is not meant to imply limitations on implementation.

With reference to Figure 5, a timing diagram for a stalling mechanism which may be employed by a shared cache controller in accordance with a preferred embodiment of the

present invention is depicted. The example depicted is derived from the 60X bus protocol. A initiation signal (TS_) driven by the processor indicates the beginning of an operation. A responsive acknowledgement signal (AACK_) driven by the L2 cache indicates that the initiation signal has been detected and is no longer required. Throttling in the depicted example may be achieved by simply delaying assertion of this responsive (AACK_) signal.

Under prior art designs, management of interactions between architectural operations and cache operations make implementing shared cache controllers extremely complicated. Implementing a shared cache controller for the prior art approach typically requires associative comparators, queue comparators, and other architectural management functions to avoid collisions, tremendously increasing the design complexity.

The layered controller of the present invention, however, scales for additional processors. Referring again to Figure 4, the actual cache 418 is only influenced by the generic interfaces 440, 444, and 428 to cache controller 416. Separate architectural controllers, noncacheable controllers, and processor-side bus interface units are required for each processor sharing cache 418, but additional processors may be supported by the cache controller--and therefore by a single shared cache--by simply adding another instance of the generic interfaces 440 and 444, and appropriately modifying throttle unit 450. The bus interface units, designed to support multiple instances of the generic interfaces, are easily expandable and need not be functionally altered beyond duplication of functional units within a BIU to provide the

required resources. Replication of the noncacheable controller, which only handles cache operations, is actually optional. Besides implementing additional instances of functional units, the only further change to cache controller 416 which might be required over the design of cache controller 212 depicted in Figure 2 is to increase the size of cache 418. A cache size suitable for nonshared implementation may not be suitable for a shared cache. Changing cache size is a fairly straightforward issue not comparable to implementing additional interlock logic.

Referring to Figure 6, a high level flowchart for a process of managing operations in a shared cache controller in accordance with a preferred embodiment of the present invention is illustrated. The process begins as step 602, which depicts detection of an operation to be performed. The process next passes to step 604, which illustrates identifying the type of the operation to determine which controller--cache, noncacheable, or architectural--should handle the operation.

The process next passes to step 606, which depicts a determination of whether the appropriate controller is available to handle the operation. If so, the process proceeds directly to step 610, depicted below. If not, the process proceeds to step 608, which illustrates stalling the operation as necessary, until the appropriate controller becomes available. The process then passes to step 610, which depicts passing the operation to the appropriate controller, and then to step 612, which depicts the process becoming idle until another operation is detected. Those skilled in the art

will appreciate that the process depicted is simplified for the purposes of explanation.

Stalling operations as described above has obvious impact on overall performance. Such performance trade-offs are acceptable, however, in view of the ease within which a controller design may be extended from nonshared to shared. Design extension may be particularly important due to short product life cycles.

The present invention facilitates extension of a nonshared cache controller design to a shared cache controller design by layering cache and architectural specific functions. Because cache operations are isolated from the variable timing associated with architectural operations, one cache operation within a colliding pair of cache operations may be briefly delayed to serialize the operations and regulate the operation flow rate. This allows a cache controller design for a nonshared cache to be reused in a shared cache controller with only straightforward modifications.

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

THIS SHEET HAS BEEN
REPLACED



CLAIMS

1. A method of managing operations in a shared cache controller, comprising:

for each operation detected, identifying a controller within a plurality of controllers to which an operation should be passed;

responsive to detecting a collision between a plurality of operations initiated by different processors, selecting an operation within said plurality of operations; and

stalling said selected operation until completion of another operation within said plurality of operations, wherein operations from a plurality of processors are serialized and regulated for said shared cache controller.

2. A shared cache controller, comprising:

a plurality of controller units including controller units for different types of operations; and

a throttle unit serializing a plurality of operations of a type handled by a single controller unit within said plurality of controller units, said plurality of operations initiated by a plurality of devices, wherein a flow rate of operations to said shared cache controller is regulated.

3. The shared cache controller of claim 2, wherein said throttle unit serializes said plurality of operations by stalling at least one operation within said plurality of operations.

This Sheet Has Been Replaced

