

**METHOD OF LAYERING CACHE AND ARCHITECTURAL
SPECIFIC FUNCTIONS TO PERMIT GENERIC INTERFACE DEFINITION**

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 for caches in data processing system. Still more particularly, the present invention relates to layering cache and architectural specific functions within a cache controller for a cache in a data processing system to facilitate definition of generic interfaces which may be

translated to specific bus architectures.

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 5. For example, a cache controller 502 may include logic 504 for maintaining the cache directory, logic 506 for implementing a least recently used (LRU) replacement policy, logic for managing reload buffers 508, and logic for managing store-back buffers 510. 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.

In addition to being processor-specific, cache controllers are generally also bus-specific. That is, cache controller implementations are typically specific to protocols governing transactions on the bus between the processor(s) and the cache controller(s). In fact, actions within the cache controller are initiated by bus operations. Thus, the cache controller is closely coupled in both design and operation to the specific buses connected to the cache controller.

Processors employed in data processing systems of the type discussed generally support a significant number of cache management instructions, such as a flush command, data cache block store, etc. Support for these instructions is typically

implemented over the cache logic, substantially complicating the cache controller design requirements by imposing special cases for different instructions.

One problem with the prior art approach to implementing cache controllers is the virtual impossibility of defining a generic interface between the cache controller logic and a bus interface unit to a bus. Such a generic interface would be useful since it would allow the cache controller logic to be reused when revising the overall cache controller design for operation with a different processor or bus architecture. Under the prior art approach, the cache controller logic cannot be simply reused independent of any bus interface unit within the cache controller.

It would be desirable, therefore, to implement a cache controller design which permits definition of a generic interface between the cache controller logic and a bus interface unit for the cache controller. It would also be advantageous if the cache controller design simultaneously facilitated reuse of the cache controller logic. Finally, it would be advantageous for the cache controller design to permit implementation of generic interfaces which may be customized depending on the characteristics of particular bus operations.

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 generic interface between the cache controller logic and a bus interface unit within the cache controller.

The foregoing objects are achieved as is now described. Cache and architectural functions within a cache controller are layered and provided with generic interfaces. Layering cache and architectural operations allows the definition of generic interfaces between controller logic and bus interface units within the controller. The generic interfaces are defined by extracting the essence of supported operations into a generic protocol. The interfaces themselves may be pulsed or held interfaces, depending on the character of the operation. Because the controller logic is isolated from the specific protocols required by a processor or bus architecture, the design may be directly transferred to new controllers for different protocols or processors by modifying the bus interface units appropriately.

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

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 within which a preferred embodiment of the present invention may be implemented;

Figure 2 is a block diagram of a controller within a data processing system in accordance with a preferred embodiment of the present invention;

Figure 3 depicts timing diagrams for variants of generic interfaces within a controller in accordance with a preferred embodiment of the present invention;

Figure 4 is a high level flowchart for a process of employing generic interfaces within a controller layered into cache and architectural specific functions in accordance with a preferred embodiment of the present invention; and

Figure 5 depicts a block diagram of a prior art cache controller.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures, and in particular with reference to Figure 1, a block diagram of a data processing system in which a preferred embodiment of the present invention may be implemented 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 combined level two (L2) cache 108. 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 controller within a data processing system 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. It is virtually impossible, however, to design generic interfaces without layering cache and architectural specific functions to separate complexities.

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 for different processors. The cache controller logic may be reused without modification for the 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.

Bus interface units 204 and 208 are responsible for managing transactions and protocols to bus 206 and system bus 210. If generic interfaces are defined for interfaces 220-230, the designs for controllers 212, 214, and 218 may be isolated from specific bus architectures. Bus interface units 204 and 208 translate the specific bus transactions into the protocol for the generic interfaces 220-230. Controllers 212, 214, and 218 are then independent, insulated from changes in bus protocols or processors.

In defining the generic interfaces, the interfaces from a given bus interface unit 204 or 208 to each of controllers 212, 214, and 218 will differ, especially when bus 206 differs from 210. Similarly, interfaces between any specific controller 212, 214, or 218 to each bus interface unit 204 and 208 will differ. Thus, each set of interfaces 220-230 will be different. The interfaces are generic because they are designed to handle any operation outside the BIUs by translating the operation to a generic protocol. The

essential essence of the bus operation is extracted from the bus protocol and communicated to the controllers by the BIUs through the generic interfaces.

To define generic interfaces 220-230, a generic protocol must be extracted from the operations passed through the interface. Regardless of differences in bus or processor architectures, common threads may be ascertained within equivalent operations. READs and WRITEs, for example, always work in essentially the same manner, but perhaps with timing or duration differences that are hidden by the BIUs or managed in the details of the individual generic protocols.

With reference now to Figure 3, timing diagrams for variants of generic interfaces within a controller in accordance with a preferred embodiment of the present invention are depicted. For a "pulsed" interface, the bus interface unit is responsible for gathering all of the information relevant to an operation and packaging that information into a packet. The bus interface unit then transmits a ready signal (RDY) and sends the information for the operation in a single packet. Pulsed interfaces are best suited for operations in which all of the relevant information becomes available quickly, in a relatively short span of time. The collected information is transmitted efficiently, often in a single cycle. A noncacheable STORE is an example of an operation which is suitable for a pulsed interface, since the processor will quickly produce all the information required for the operation.

A "held" interface typically involves a request signal (REQ) which is asserted for some period of time. Information

relevant to the requested operation--qualifying information such as address or field codes--has been previously assembled by the source. A variety of other signals are exchanged between the source and the target to control the data transfer. A done signal (DONE) terminates the exchange. Held interfaces are best suited for operations which have an unbounded nature, such as those which may require a long and potentially variable amount of time. An example is a READ operation, which can take an arbitrary amount of time that cannot be controlled.

Those skilled in the art will recognize that other interfaces besides the pulsed and held interfaces described will be suitable for implementing the generic interfaces within a controller layered into cache and architectural specific functions, including variations of the interfaces described. For example, instead of an explicit DONE signal, the signal may be implicit from other control signals. The most suitable interface will be driven by the character of operation for which the interface is being implemented.

Referring again to Figure 2 each of interfaces 220-230 is actually a group of interfaces. Interface set 220, for example, includes at least one distinct interface for each operation which will be passed between BIU 204 and cache controller 212. For performance reasons, multiple replications of an interface for a specific operation may be included within a particular interface set. Interfaces for specific operations will be implemented based on the type of operation and the suitability of pulsed, held, or other interfaces as described above.

Referring to Figure 4, a high level flowchart for a process of employing generic interfaces within a controller layered into cache and architectural specific functions in accordance with a preferred embodiment of the present invention is depicted. The process may be performed in a controller layered into cache and architectural specific operations and including generic interfaces to the cache and architectural controllers.

The process begins at step 402, which depicts detecting an operation on the processor bus. The process then passes to step 404, which illustrates translation of the operation into the generic protocol employed by one of the generic interfaces, and to step 406, which depicts a determination of whether the instruction is a cache operation or an architectural operation. Although depicted separately, steps 404 and 406 will overlap since the nature of the operation must be determined to translate the operation for the appropriate interface.

If the operation is a cache operation, the process proceeds to step 408, which illustrates performing the operation in the cache controller. The process then passes to step 416, which depicts the process becoming idle until another operation is detected. Referring again to step 406, if the operation is not a cache operation the process proceeds instead to step 410, which depicts parsing the operation into cache operations, and then to step 412, which depicts translating the cache operations into the system bus protocol. Steps 410 and 412 may overlap.

The process then passes to step 414, which illustrates

transmitting the cache operations on the system bus. The process next passes to step 416, which depicts the process becoming idle until another operation is detected.

The process described relates to operation of a bus interface unit on the processor side of the controller. A similar process occurs on the system bus side of the controller, but requiring only steps 402, 404 and 406.

The present invention provides the ability to insulate controller logic designs from changes in protocols and/or processors by allowing generic interface to be utilized in connection with layered cache and architectural operations. This is not possible under the prior art approach of overlaying the semantic and temporal complexities of architectural operations with simpler cache operations. Architectural operations exhibit an irregular temporal behavior, with the same operation requiring a variable amount of time in different instances. Architectural operations also typically require support for special cases. The more interrelated architectural and cache operations are, the more difficult it becomes to define generic interfaces and achieve the associated benefits.

The present invention also divides complexities between appropriate segments of the controller, moving them to appropriate places. Each segment of the controller thus has a well-defined job within a well-defined interfaces. This divide-and-conquer approach layers design efforts into more manageable, relatively autonomous units.

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 employing generic interfaces in a cache controller, comprising:

receiving a bus operation in a specific protocol;

translating said bus operation into a protocol employed by a generic interface for a controller logic unit within a plurality of controller logic units, each controller logic unit within said plurality of controller logic units handling a different class of operations; and

transmitting said translated bus operation to said controller logic unit for performance, wherein generic, architecture independent interfaces may be utilized for controller logic units layering classes of operations within a cache controller.

2. A cache controller, comprising:

a bus interface unit;

a plurality of controller logic units, each controller logic unit within said plurality of controller logic units handling a different class of operations; and

at least one generic interface between a controller logic unit within said plurality of controller logic units and said bus interface unit, wherein controller logic units layering classes of operations within a cache controller are provided with architecture independent interfaces.

This Sheet Has Been Replaced



* C L M S *