ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

Cache Controller with Enhanced Features using Verilog HDL

Prof. V. B. Baru[1], Sweety Pinjani[2]

Assistant Professor, Dept. of ECE, Sinhgad College of Engineering, Vadgaon (BK), Pune, India [1]

PG Student [VLSI & Embedded Systems], Sinhgad College of Engineering, Vadgaon (BK), Pune, India [2]

ABSTRACT: The major role of cache controller is reduction in the data transfer access time between the CPU and cache. The fact that read request is more critical compared to write request is exploited in this paper. The paper presents a novel approach to cache controllers which uses read write partitioning wherein more read lines are maintained as compared to write lines.

KEYWORDS: hit, miss, RWP, cache, controller.

I.INTRODUCTION

There is a drastic improvement in technology leading to improved processor speeds. However, the memory speed has not yet increased in order to match with the faster processing needs. As cache is the fastest memory, there is a need for cache controller enhancements. Some methods for improving cache controller have been proposed in the past. One of such methods is to design a pipelined cache controller which increases the circuit complexity. Another method is to increase the size of the cache. But, as the cache size increases, the cache access time also increases. So, there is a need of better and efficient techniques.

Multi-core processor is the modern processor which can handle multiple applications simultaneously. Multiple cores on a single die share the cache resources in such processors. So, there is a need for the hierarchical cache with different cache levels like L1, L2 and LLC. Thus, the cache controller must be capable of handling hierarchical cache in the multi-core processor. Some prior works have proposed various insertion policies or algorithms for storing data inside the cache. Some attempted to differentiate between the “critical” and “not-critical” data. Some others focused on the likelihood of processor stall as the main criteria. Prior work focused only on distinguishing the lines that will be reused and those that will be not. It failed to distinguish between lines reused by reads versus those reused by writes. The criticality of read requests is more as compared to write request. This is due to the fact that write data can be temporarily stored inside a buffer. Hence, read must be favored compared to a write request. The technique proposed in this paper makes this distinction between read-write criticality.

The controller design proposed in this paper applies multiple ways and block size to the cache dynamically. In order to further improve the system performance, the controller is designed to be capable of handling the multi-sized output. The cache management policy used increases the read request hit probability by causing less critical write requests to miss. It also distinguishes between the cache lines which will be read in future and those that will be write only. An attempt is made to increase the cache lines that will be read and eliminate those that will be write only.

The contributions made in this paper are:

- Design a controller which can be used for multi-core processor where multiple cores share the cache.
- Capability to handle the multiple ways and block size for cache and provide multiple-sized output.
- Distinction between read and write requests criticality.
- Read Write Partitioning (RWP) is proposed as a cache management policy.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

II.LITERATURE SURVEY

The cache is replacement policies are optimizing instructions in order to manage a cache of information stored on a system. When the cache is full, the algorithm must choose which items to discard to make room for the new ones. Several such policies have been proposed in the past which are described below.
The Least Recently Used (LRU) discards the least recently used items first. The algorithm need to track the past access history of the data maintained inside the cache. General technique used for this purpose is the use of “age bits” for cache lines. Every time a cache line is used, the age of all other cache lines changes.

If CPU caches have large associativity (>4ways), the implementation cost of LRU becomes prohibitive. A scheme that discards one of the least recently used items is sufficient. So, CPU designers choose a Pseudo Least Recently Used (PLRU) algorithm which only needs one bit per cache item. It typically has a slightly worse miss ratio, better latency and uses less power than LRU.

The Most Recently Used (MRU) is another cache replacement policy which discards the most recently used items first. It is the best replacement policy when a file is being repeatedly scanned in a reference pattern. MRU has more hits than LRU for random access patterns and repeated scans over large datasets. MRU has a tendency to retain old data. They are most useful in situations where the older item is more likely to be accessed.

Random Replacement (RR) selects a candidate item randomly and discards it to make space when necessary. It does not need to store any information about the access history. It is relatively simple and admits stochastic simulation. Least Frequently Used (LFU) involves the system keeping track of the number of times a block is referenced in memory. When the system is full, the algorithm will replace the item with the lowest frequency.

III.THE CACHE

The cache is the fastest and the most expensive memory. It is the highest part in the memory hierarchy tree.

A. *The Architecture*

The cache implemented in this paper is a dynamic one. It is having a memory size of 4 MB and a hierarchical cache with L1, L2 and LLC. It may be implemented as a 16, 8, 4 or 2 way set associative cache. Also, the block size may vary from 16, 8, 4 or 2 bytes. The number of sets for each way and the data handling capability for each set will vary according to the way and block size chosen for the cache.

B. *Operation*

Two different operations are possible for the cache. These are to read the data from cache or to write the data inside the cache. They are explained below.

1) Write Operation

Whenever there is a write request from the processor, the way and the cache line for cache write will be selected by RWP cache management policy. Only one way which is selected by RWP is enabled for write operation. The data needs to be converted according to the data alignment of the cache before performing this operation.

2) Read Operation

All ways are enabled for cache read operation. The set address decides which set is to be accessed. The tag address is then compared using a comparator inside each way. If the tag address matches and the valid bit is set, it is an indication for the availability of data inside the cache. This is termed as *hit*. The way which asserted the *hit* is selected by the data multiplexer. The data is selected via data encoder. There will be only one *hit* at a time. If the tag address does not match, it is a *miss* indicating the data is not present inside the cache and no data will be provided by cache. The data inside the cache will be 16, 8, 4 or 2 byte. This data needs to be converted into 8 byte data as will be accepted by the processor.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

IV.THE CACHE CONTROLLER

The main function of the cache controller is to increase the data access speed between cache memory and the processor. Whenever the controller receives the request from the processor for data access, it will check if the data is present inside the cache. If the data exists inside the cache, the controller will send the data from cache to the processor. However, if the data is not present inside the cache, the controller will fetch the data from the main memory and send a copy of data to the processor and to the cache according to the cache management policy used.

A. The Architecture

The controller will have stages like fetch data, read cache, read main memory, write cache, write main memory and provide data to the processor. These stages are designed using FSM. The stages are again divided into different states as shown in Figure.1.

The controller is capable of handling the caches with multiple ways like 16, 8, 4 or 2 ways as well as various block size like 16, 8, 4 or 2 byte. Also, it is capable of delivering the data as multiple-sized output like 16, 8, 4, 2 or 1 byte. One of the major challenges for the cache controller is to align the variable data exchange between the processor, the cache and the main memory.

The controller is designed to deal with three different agents or processors. In order to overcome the data transfer clashes, the controller will use a *busy* flag. Whenever the cache is being accessed the *busy* flag will be asserted. The processor can request data only if the busy flag is not asserted.

B. Operation

The controller consists of various states: *Fetch Data, Read Cache, Read Memory, Write Cache, Write Memory and Give Data.*

1) *Fetch Data:* At this state, the controller will check whether the request from the processor is read or write. This can be analyzed on the basis of the instruction opcode. The controller will maintain the *busy* flag low until it goes to any next state.

2) *Read Cache:* At this state, the controller will check the cache for the availability of the requested data. If the data is present inside the cache, it is a *hit* and the requested data is sent to the processor. Thus, the controller will go to the next state *Give Data.* This state also has other states for data alignment. If the data is not present inside the cache, it is a *miss* and the controller will now search for the data inside the main memory. The controller will then go to the next state *Read Memory.*

3) *Read Memory:* At this state, the main memory is checked for the availability of the requested data. Then one copy of the data is sent to the processor and the other one is sent to the cache according to the RWP cache replacement policy used by the controller. This state also involves the conversion of the data from the main memory to a 16, 8, 4 or 2 byte data as will be acceptable to the cache. It many involve a parallel to serial converter where multiple smaller data is combined into one big data or it may involve the serial to parallel converter where one big data is split into several small data.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

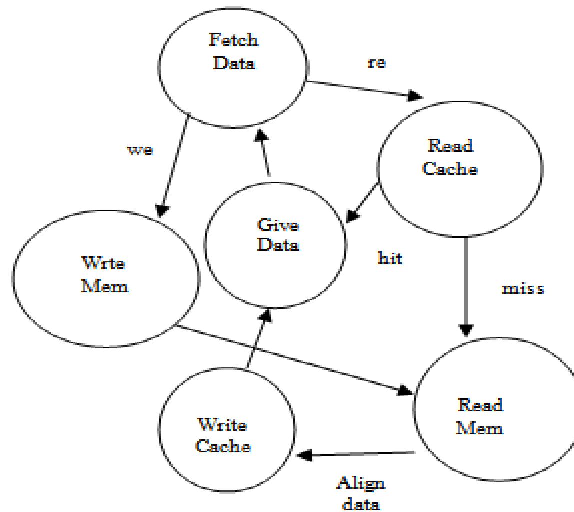

Figure.1. State diagram of cache controller

4) *Write Cache:* At this state, the controller will write the data inside the cache. The cache line for writing the data will be selected with the help of RWP cache management policy. There are two conditions in which this state may be accessed. If this state is accessed due to cache *miss*, the controller will return back to the *Read Cache* followed by the *Give Data* state. If the state is accessed due to the write request sent by the processor, the controller will go to the *Fetch Data* state in order to fetch the next instruction from the processor. This state also needs some more additional states for data alignment between the processor, the cache and the main memory as will be required.

5) *Write Memory:* In this state the data from the processor is written to the main memory. This state occurs only if the processor requests from the write operation. This data may then be stored into the cache if it has a probability for read in future.

V.THE CACHE MANAGEMENT POLICY

This paper uses the Read-Write Partitioning (RWP) policy which distinguishes the criticality of read and writes, giving a higher priority to read. This is achieved by maximizing the hit probability for read at the cost of minimizing that hit for the write, if required. In order to distinguish between the cache lines which will be read and those that will be write in future, the cache lines are logically partitioned into two categories as described below:

1) *Clean lines*: These are the lines which will be read at least once before being evicted.

2) *Dirty lines*: These are the lines used for write, independent of the probability of future read request. Write-only lines, a subset of dirty lines are those lines that will be evicted without any read.
Structure for a RWP is as shown in the Figure.2. Partition Size predictor predicts the best partition size for the clean and dirty partitions. In order to service the write request, either a clean or a dirty line will be evicted. The evicted line will be replaced by the new line.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

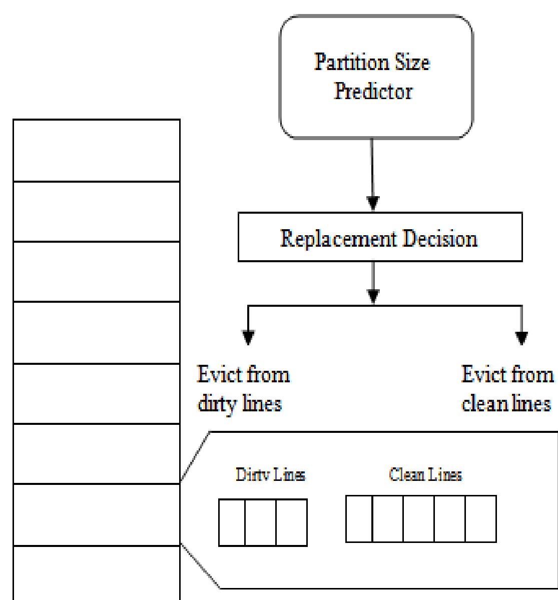

Figure.2. Structure of RWP

VI. THE RWP FRAMEWORK

RWP framework partitions the cache into clean and dirty lines logically, not physically. In order to indicate this, a dirty status bit is used. Whenever a write is made to the clean line, its dirty bit is set and it logically becomes a part of the dirty partition. There is no strict enforcement on the partition sizes. They are adjusted only when a new cache line is allocated. RWP dynamically determines the cache line to be evicted and replaced with the new one. There can be three possible cases to determine the placement victim based on the current number of dirty lines in the cache set. They are described below:

- The current dirty lines are greater than the best predicted partition size: RWP picks the LRU line from the dirty partition as the replacement victim.
- The current dirty lines are smaller than the best predicted partition size: RWP picks the LRU line from the clean partition as the replacement victim.
- The current dirty lines are equal to the best predicted partition size: The replacement victim depends on the access type of the request. If the access type is read, RWP picks the victim from clean partition. However, for a write access, the replacement victim is picked from the dirty partition.

RWP calculates the best partition sizes dynamically and makes a conclusion on the number of ways to be assigned for each partition. In order to achieve this, it needs to calculate the benefit of growing one partition size at the cost of shrinking another one. RWP compares the read reuse of clean and dirty lines as if given exclusive access to the entire cache.

The Partition Size Predictor uses set sampling, wherein a small subset of total sets is extended with a shadow directory. Whenever there is a read miss, a cache line is allocated to the clean shadow directory. For a write miss, the cache line is allocated to the dirty shadow directory. Two counters are used for size prediction: clean age hit counter and dirty age hit counter. If a read request is made to the clean partition, clean age hit counter is incremented by 1. Similarly, if the read request is made to the dirty partition, dirty age hit counter is incremented by 1. These counter values are used to predict the number of additional hit/miss that cache will incur if one of the ways allocated to a partition is re-allocated to another. Thus, the best partition size is predicted.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

VII.CONCLUSION

The criticality of read requests is more as compared to the writes. Thus, the cache management policies that increase the probability of cache hits perform better. This is done by allocating more resources to the reads. RWP is a mechanism which distinguishes between read lines vs. write-only lines and also favors read lines. It protects the partition with read lines and evicts the partition with write-only lines. Thus, RWP can outperform many of the prior cache management mechanisms.

REFERENCES

[1] Samira Khan and Chris Wilkerson, "Improving Cache Performance Using Read-Write Partitioning", IEEE, 2014.
[2] Siti Lailatul Mohd Hassan, "Multi-Sized Output Cache Controllers", International Conference on Technology, Informatics, Management, Engineering & Environment, pp. 186-191, 23-26 June 2013.
[3] C. J. Lee, DRAM-aware last-level cache write back: Reducing write-caused interference in memory systems, Technical Report TR-HPS-2010-002, pp. 1-25, December 2010.
[4] M. Qureshi et al. Improving read performance of phase change memories via write cancellation and write pausing, HPCA, 2010.
[5] M. Chaudhuri, Pseudo-LIFO: The foundation of a new family of replacement policies for last-level caches, MICRO, 2009.