IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

Implementation of MESI Protocol Using Verilog

Penmetsa Durga Devi[1], Tammineni Ravindra[2]

M.Tech Student, ECE, GVPCOE (A), Visakhapatnam, India[1]

Assistant Professor, ECE, GVPCOE (A), Visakhapatnam, India[2]

Abstract: Multiprocessor system is a system which contains two or more processors working simultaneously and sharing the same memory. Nowadays multiprocessors are being widely used due to their high throughput and reliability. It is important to maintain data consistency in multiprocessor system as different processors may communicate and share the data with each other. In multiprocessor systems caching plays a very important role. Cache coherence is a major issue in multiprocessor systems. In this paper we have designed three direct-mapped caches and to maintain cache coherence and data consistency among the processors we have used the MESI protocol. The MESI protocol is invalidation based cache coherence protocol. In this protocol each cache block can be in one of four states i.e., Modified, Exclusive, Shared and Invalid. In this protocol, whenever a processor writes into the local cache, all copies of it in other processors are invalidated in order to maintain data consistency and cache coherence. The cache design is simulated and synthesized using Xilinx ISE 14.7 Simulator and XST Synthesizer.

Keywords: Cache Coherence, Direct-mapped Cache, MESI, Xilinx.

I. INTRODUCTION

In recent years multiprocessors are gaining more importance as they have better performance and reliability than single processor systems. Multiprocessors with shared memory are being used in the today's computers and researches [1]. Using the single address space the processors can communicate among themselves because address space is shared among the processors in multiprocessor systems. So, same cache entry exists in other processors as the address is being shared. The shared memory multiprocessor system architecture is shown in the Fig.1.Sharing of data among the processors is not a problem during reading operation but it is a serious problem during write operation. When one processor writes a value to a location that is being shared, the changed value has to be updated to all caches otherwise the processors hold different data for the same location which is called as cache coherence problem [2].

Consider three processors named P1, P2, P3 sharing a same memory [3]. The processor P1 wants to read a value at location X. It reads the value from main memory and caches its value into P1. The processor P2 wants to read a value at location X. It reads the value from main memory and caches its value into P2as shown in Fig.2. The processor P1 wants to write a value to location X. This is shown in Fig.3. Now if processor P3 performs read operation on location X. In p1 at location X, value stored is 10 and in p2 at location X, the data stored is 5, therefore data inconsistency arises when we perform write operation to a shared address location. This is shown in Fig.4.

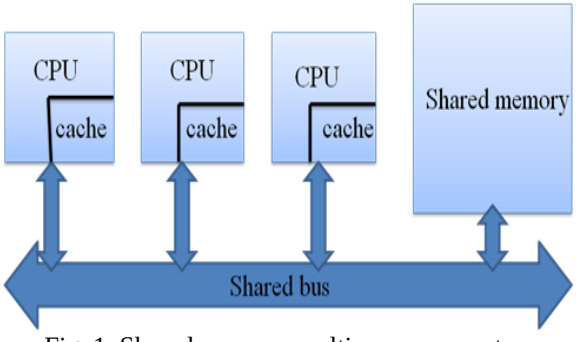

Fig. 1. Shared memory multiprocessor system

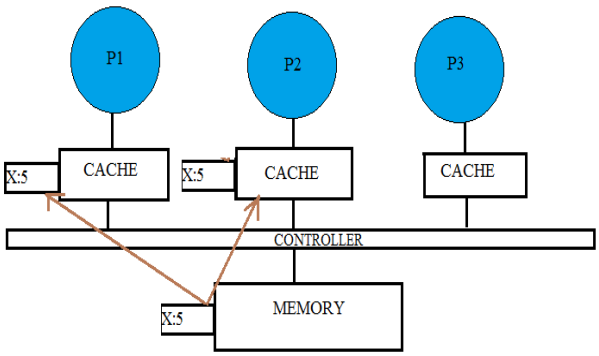

Fig. 2. P1, P2 reading values from memory

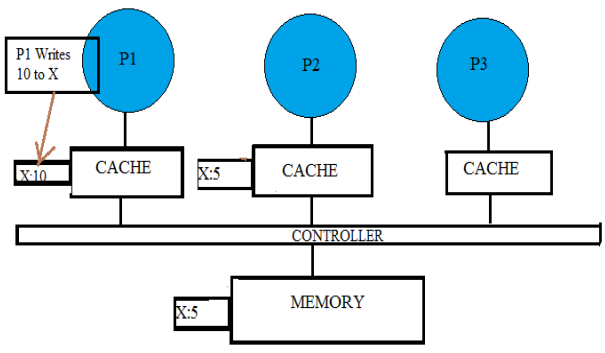

Fig. 3. P1 performing write operation in location X

DOI 10.17148/IJARCCE.2016.51046

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

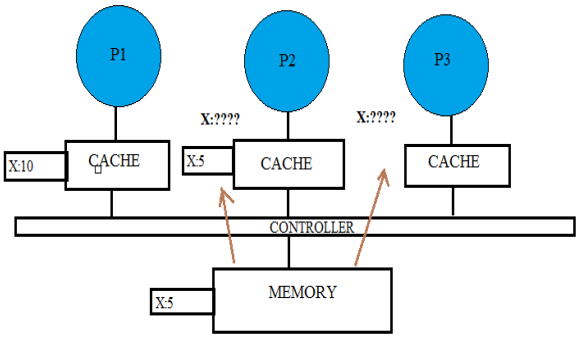

Fig. 4. Data inconsistency in multiprocessor systems

The paper objective is to overcome the data inconsistency and cache non-coherence in multiprocessor systems. To obtain data consistency and cache coherence in multiprocessor systems, MESI Protocol [4-6] is implemented in this thesis work. In MESI protocol, whenever a write operation to a shared location is performed, an invalidate signal is issued by the cache controller to all those caches containing same address location as shown in the Fig.5, so that no stale data is read by the processors.

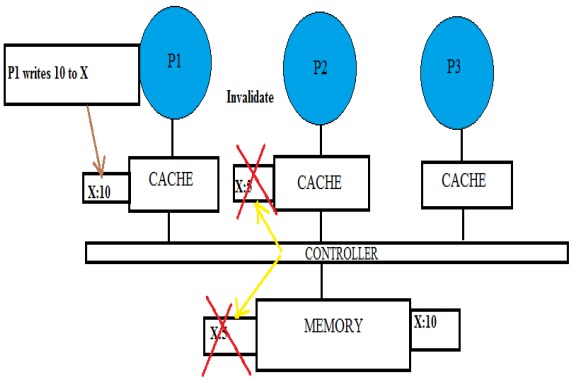

Fig. 5. Cache coherence in multiprocessor systems using MESI Protocol

II. CACHE MEMORY

Cache memory is being used in the modern computer systems, to temporarily hold the currently accessed contents of the main memory locations[7]. To the cache the processor sends four input signals, they are cpu_cac_add, cpu_cac_read, cpu_cac_data and cpu_cac_wrt. To the processor the cache sends three outputs they are cac_cpu_hit, cac_cpu_miss, cac_cpu_data. If the processor requested address location is not found in cache then the cache sends cac_mem_read, cac_mem_add, cac_mem_data and cac_mem_wrt as inputs to the memory. After data is fetched from memory, the data is placed in the cache through mem_cac_data signal sent by the memory to the cache as shown in the Fig.6.

The memory that is designed in this paper is of 32bytes. Therefore five address bits are required. Eight bytes size direct mapped cache is designed in this paper.

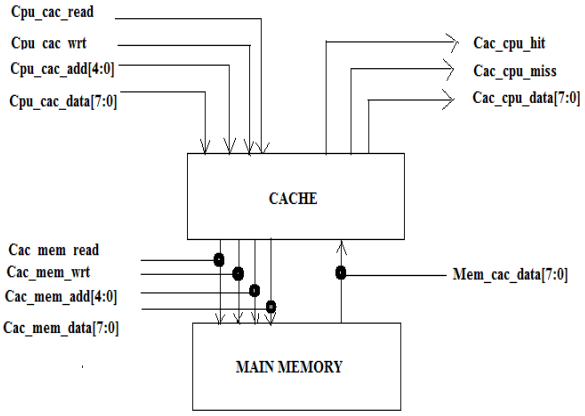

Fig. 6. Architectural view

III. DIRECT MAPPED CACHE DESIGN

In this cache organization, each location in main memory can go in only one entry in the cache. One way set associative cache is the other name for direct mapped cache [8-9]. There is no replacement policy like least recently used algorithm etc., in direct mapped cache because there is only one cache entry. This is the simple form of cache.

A. Cache entry structure

Cache row entries usually have the following structure:

Dirty(1 bit)	Valid(1 bit)	Tag(2 bits)	Data(8 bits)

Fig. 7. Cache entry structure

- Dirty bit: The dirty bit is set, whenever a write operation is performed on cache. The writing policy that we have used in this paper is write-back policy. In this policy writes are not immediately reflected to main memory.
  In this whenever a write operation is performed on cache, it is marked dirty. If the cache line marked dirty is evicted from cache block, the changed data is written to that particular location in main memory and the dirty bit is made to zero.

- Valid bit: The valid bit is used to know whether the data present in the cache entry is valid or not. If the data present in the cache entry is valid, then the valid bit is set otherwise it is cleared.

- Tag: The part of the cache line that stores the address of the block is called the tag field. Tags are added to the cache entry along with data in order to supply the remaining bits of address, which are used to differentiate memory locations that are mapped to the same cache block.

The CPU address is divided into index and tag. 32 bytes of main memory is taken, so the 5 address bits are divided into tag and index. The MSB 2 bits are tag bits and the remaining 3 bits are index bits. This is shown in the Fig. 8.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

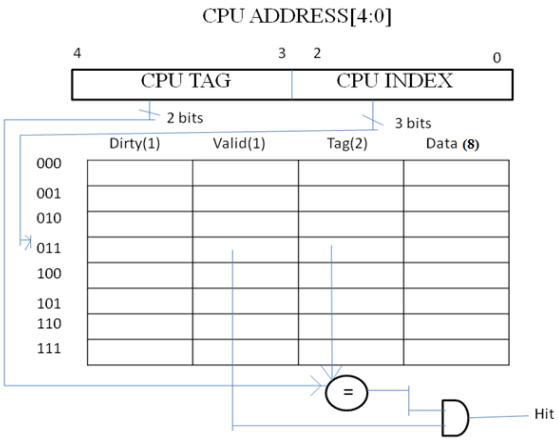

Fig. 8. Cache hit

B. Operations performed on cache

Four operations are performed on cache[10]. They are:

- Read hit: If the cpu_tag and the tag stored inside the cache match with each other, then read hit is said to occur.
- Read miss: If the cpu_tag and the tag stored inside the cache does not match with each other, then read miss is said to occur. As the required address is not present in the cache, the processor sends the address and read signal to the memory and after fetching the data, the cache is updated with this data.
- Write hit: If the cpu_tag and the tag stored inside the cache match with each other , then write hit is said to occur and the dirty bit is set.
- Write miss: If the cpu_tag and the tag stored inside the cache does not match with each other, then write miss is said to occur. As the required address is not present in the cache, the processor sends the address and write signal to the memory and after writing the data, the cache is updated with this data.

IV. MESI PROTOCOL

MESI Protocol is the commonly used invalidation protocol. The processor's cache line can be in any of the four states [11-12]. The four states are Modified, Exclusive, Shared and Invalid.

- Modified: This is the only copy in cache. The data present in the cache is different from the data present in the main memory. So, to indicate the data change in cache, we are using a dirty bit. The dirty bit is set whenever the data in the cache is changed. After writing back the changed value to main memory, the dirty bit is cleared.
- Exclusive: This is the only copy in the cache. The data present in the cache is same as the data present in the main memory. The data present in the cache is a valid data.
- Shared: This copy is present in more than one cache. The data present in the cache is same as main memory. The data present in the cache is the valid data.
- Invalid: If the requested location is not found in cache then, the cache line is in invalid state. The cache line is also in invalid state, if the data present inside the cache line is not valid.

A. Operations performed on local caches:

1. Read hit
2. Read miss
3. Write hit
4. Write miss

The cache controller takes the following different actions based upon the read or write signals sent by the CPU.

Read Hit:

The processor sends the read signal and address to the cache, if the location is found in the cache then read hit is said to occur. The cache line can be in any one of the three states i.e., modified, exclusive and shared.

If the location is found in the cache then based on the states the controller takes the following actions:

a) M state: The controller returns the modified data to the processor and there is no state change.
b) S or E states: The controller simply returns the value and there is no state change.

Read Miss:

The processor sends the read signal and address to the cache, if the location is not found in cache then read miss is said to occur. Now the controller sends the read signal and address to the cache2, if the location is found in the cache2 then the controller puts this data to cache1 and reads the value. If the location is not found in cache2 also, then controller now sends the read signal and address to the cache3. If the location is found in cache3, then that data is placed in cache1. If the location is not found in cache3 then the controller sends read signal and address to main memory. After reading the value from memory, the data is placed into cache1.

If the location requested by the processor is not found in cache1, controller sends the read signal to remaining caches. If location found in other caches then based on the states the controller takes the following actions:

a) One cache has E copy: The controller puts this value to the cache1 and changes both the cache line states to shared state.
b) Several caches have S copy: The controller puts this value to the cache1 and changes cache1 state to shared state and leaves other states state to shared state.
c) One cache has M copy: The controller puts this value to cache1 and marks its state as shared. Now the other cache modified value is copied back to memory and its state is changed from modified to shared state.
d) No other caches have the copy: The controller reads the value from memory and puts this value into cache1 and cache1 state is marked as exclusive as this the only cached copy

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

Write Hit:

The processor sends the write signal and address to the cache, if the location is found in the cache, then write hit is said to occur. After writing the new data to the cache, the dirty bit is set. The dirty bit is cleared after writing back this modified data to the main memory. During write operation, if the cache line is in shared state then the controller changes its other copies state to invalid state.

If the location is found in the cache then based on the states the controller takes the following actions:

a) M state: The controller updates the local cache and sets the dirty bit. The controller doesn't change the state.
b) E state: The controller updates the local cache and controller changes its state from exclusive to modified.
c) S state: The controller updates the local cache and controller changes the local cache state to modified and the controller changes other caches states from shared to invalid.

Write miss:

The processor sends the write signal and address to the cache, if the location is not found in the cache, then write miss is said to occur.

a) No other copies in caches: The controller writes the value to memory and it places this value to local cache and marks its state as modified.
b) One cache has E copy or more caches have S copy: The controller reads the value from memory and local copy is updated and state is set to modified state. The controller invalidates the other cache copies and changes their state to invalid.
c) One cache has M copy: The controller after writing back its value to memory its state is marked as invalid. The local copy value is updated and the state is set to modified.

V. SIMULATION RESULTS

In this paper the results are simulated using Xilinx ISE 14.7 Simulator.

A. Simulation results for single cache

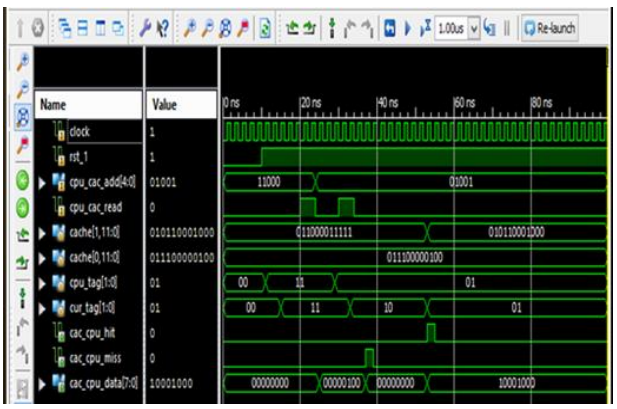

Fig. 9. Cache Read hit and Read Miss

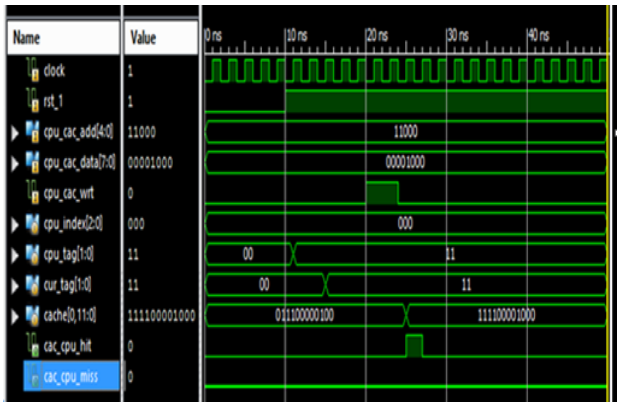

Fig. 10. Cache Write hit

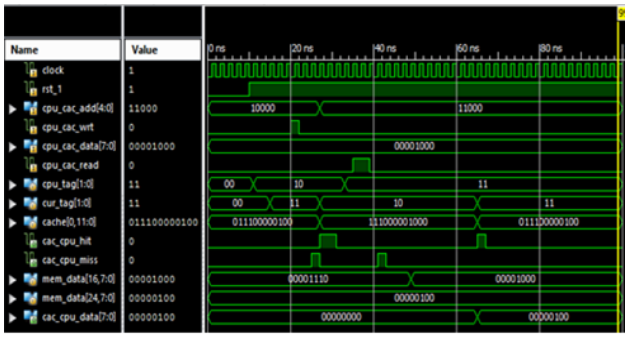

Fig. 11.Cache write back mechanism

B. Simulation results for cache coherence in multiprocessors using MESI Protocol.

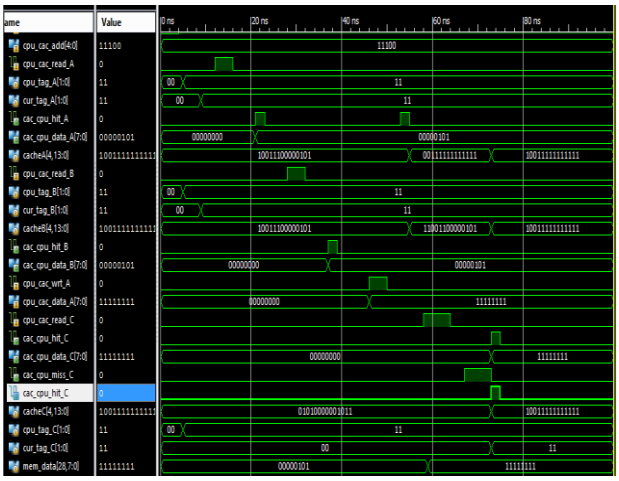

Fig. 12. Cache coherence in multiprocessor system

VI. CONCLUSION

In recent years, multiprocessor systems are being widely used. Multiprocessor systems are highly reliable and economic when compared to uniprocessor systems. Sharing of data among the processors is not a problem during the read operation but it is a serious problem during write operation. When one processor writes a value to a location that is being shared, the changed value has to be updated to all caches otherwise, the processors hold different data for the same address. The major issue in

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

multiprocessor system is data inconsistency and cache non-coherence. To overcome these drawbacks a MESI cache coherence protocol is implemented in this paper. The MESI is a four state, invalidation based protocol. We have designed three caches and have implemented MESI Protocol, to achieve cache coherence and data consistency.

REFERENCES

[1] Kalyani D. Kohle, U. M. Gokhale, Darshan Pendhari, "Design of cache controller for multicore systems using parallelization method", Proceedings of 11th IRF international conference, June 2014.
[2] Dubois, Briggs,"Effects of cache coherency in multiprocessors", in IEEE Transactions on computers, vol. 31, issue-11, Nov 1982, pp. 1083-1099.
[3] David J.Lilja, " Cache coherence in large scale shared memory multiprocessors", ACM Computing surveys, vol. 25, No.3, Sept. 1993, pp. 303-338.
[4] Daniel J. Sorin, Manoj Plakal, Anne E. Condon, "Specifying and verifying a broadcast and a multicast snooping cache coherence protocol", IEEE Transactions on parallel and distributed systems, vol. 13, No. 6, June 2002.
[5] Kaushik Roy, Payan kumar, Meenatchi ,"Comparative study on cache coherence protocols", IOSR Journal of computer engineering, vol. 17,issue. 3,Ver-1,May-June 2015, pp. 71-75.
[6] Xiantuo Rao, Teng Wang, Xin'an Wang, "A Low-Power and High-efficiency Cache Design for Embedded Bus-based Symmetric Multiprocessors", 2013 IEEE conference, pages 1-4.
[7] Linda Null, Julia lobus, "The essentials of computer organisation and architecture",3rd edition, Jones and Bartlett learning.
[8] Pavan Shree B. V, Mrs. Anitha V, " Design and implementation of direct mapped cache memory with same tag bit information",International journal of computer science and mobile computing, vol. 4,issue-6, June 2015, pp. 978-983.
[9] D.A.Patterison, J.I.Hennessy.1998.Computer organization and design.4th edition.
[10] Zhenghong Wang and Ruby B. Lee," A Novel cache architecture with enhanced performance and security", 41st IEEE international symposium on microarchitecture, 2008, pp. 83-93.
[11] Sultan Almakdi, Abdul Wahab Alazeb, Mohammad Alshahari, "Cache coherence mechanisms", International journal of engineering and innovative technology, vol. 4, issue-7, January 2015.
[12] Lubomir Ivano, New Rochelle, "Modelling and verification of cache coherence protocols", IEEE international symposium on circuits and systems, vol. 5, may 2001, pp. 129-132.

DOI 10.17148/IJARCCE.2016.51046