International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 06 | June 2019 www.irjet.net p-ISSN: 2395-0072

IMPLEMENTATION OF MESI PROTOCOL USING VERILOG

Attada Sravanthi[1], Ch. Rajasekhara Rao[2], K. Krishnam Raju[3], L. Rambabu[4]

[1]PG Scholar, Aditya Institute of Technology and Management, Tekkali, 532201, India
[2,3,4]Associate, Aditya Institute of Technology and Management, Tekkali, 532201, India

---

Abstract:- Multiprocessor system has 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 multi-processor systems as different processors may communicate and share the data with each other. In multiprocessor systems caching plays a vital role. Cache coherence is a major issue in multiprocessor systems. In the present paper, three direct- mapped caches are designed and to maintain the cache coherence and data consistency among the processors, MESI protocol is used. The MESI protocol is the 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 syn papered using Xilinx ISE 14.7 Simulator and XST Synthesizer
.

Keywords: MESI

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 figure1.1.

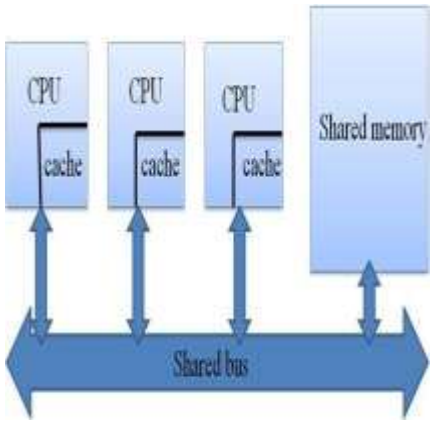

Figure 1.1: Shared memory multiprocessor system

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].

Evolution of cache coherence protocols

MI Protocol

It is the basic conventional protocol used for maintaining cache coherence in shared memory multi-processor systems. Each cache block can be in any one of the two states i.e., modified, invalid.

Modified: Modified means the data present in the cache line is different from the data present in the main memory. If the cache block is in modified state then it can perform read and write operation. Whenever a write operation is performed only one block can be in modified state and the rest should be in invalid state. Invalid: No reading and writing operations can be performed if the cache block is in invalid state. The cache block is said to be in invalid state if the required location is not present in the cache.

Advantages of MI Protocol

1) Implementation of the protocol is easier.

2) Number of transient states are less. Disadvantage of MI Protocol

1) There is no difference between shared and modified block.

MSI Protocol

It is the extension to MI Protocol. Each cache block can be in any one of the three states i.e., modified, shared, invalid [4].

a) Modified: Modified means the data present in the cache line is different from the data present in the main memory. If the cache block is in modified state then it can perform read and write operation.

b) Shared: The data present in the cache is similar to the data present in the main memory. The same location may be present in other caches also.

c) Invalid: No reading and writing operations can be performed if the cache block is in invalid state. The cache block is said to be in invalid state if the required location is not present in the cache.

Advantages of MSI Protocol

1) There is a difference between M and S states.

2) As it contains S state multiple copies of block can be present at the same time.

3) S to M transition can be made without reading data from cache.

Disadvantages of MSI Protocol

1) Whenever write request is issued, the state changes from S to M and invalidate message is sent even though it is the only copy present.

2) During a read, though there is only one copy present, the block goes to S state.

MESI Protocol

It is an extension to MSI Protocol. Each cache block can be in any one of the four states i.e., modified, exclusive, shared, invalid [5-6].

a) Modified: Modified means the data present in the cache line is different from the data present in the main memory. If the cache block is in modified state then it can perform read and write operation.

b) Exclusive: The data present in the cache line is same as that present in the main memory and it is the only cached copy.

c) Shared: The data present in the cache is similar to the data present in the main memory. The same location may be present in other caches also.

d) Invalid: No reading and writing operations can be performed if the cache block is invalid state. The cache block is said to be in invalid state if the required location is not present in the cache.

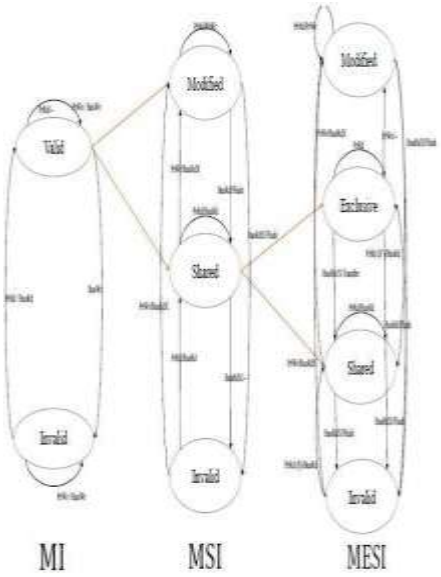

Figure 2.1: Evolution from MI to MSI and MSI to MESI

Entry contains data as well as tag as shown in the figure3.1 below.

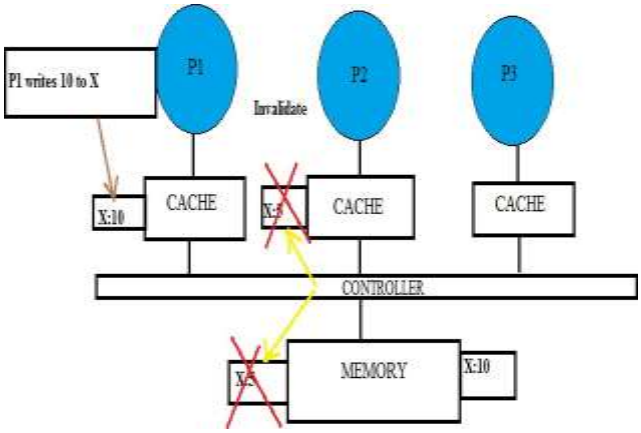

Figure 2.2 Cache coherence in multiprocessor systems using MESI protocol

Cache memory

Cache memory is being used in the modern computer systems, to temporarily hold the currently used contents of the main memory locations. Data present in the cache memory can be accessed in less time than from main memory; therefore cache memory is faster memory. Cache memory is expensive than main memory even though it is much smaller than main memory.

Cache entries

Data is transferred between memory and cache in blocks of fixed size, called cache lines. A cache entry is created, when a cache line is copied from memory to the cache. The cache

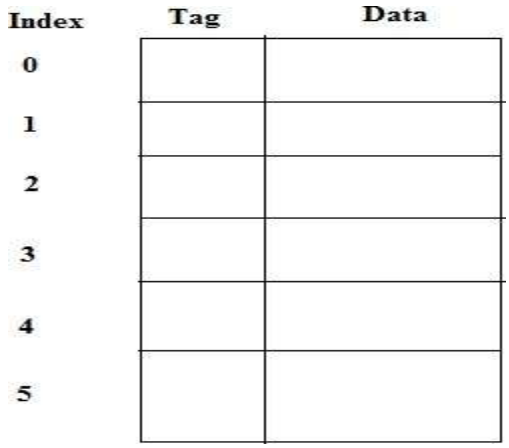

Figure 3.1: Cache entry

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. When the processor wants to read from a location or write to a location, it first checks for a corresponding entry in the cache. If the location is found in the cache then cache hit has occurred. If cache hit occurs the processor immediately reads the data from the cache line or writes the data into the cache line. If the location is not found in the cache, then cache miss has occurred, so it reads the data from main memory and copies this data into the cache.

Cache performance

To have a good cache performance, a high hit ratio and high search speed are required. The cache hit rate means the number of times hits occurred to the total number of accesses. The possibilities of cache containing memory addresses that the processor wants needs to be increased. The cache performance will be increased if more number of hits occur and decrease with more number of misses. A cache hit is said to occur, when the processor requested location is found in the cache entry. A cache miss is said to occur, when the processor requested location is not found in the cache entry. For a miss, the cache is going to get the data from memory and the data will be placed into the cache, so this takes more time and hence the performance of cache reduces.

Write policies

When a write operation is performed, the data that was written into the cache need to be copied to the main memory at some point of time. In cache there are two writing policies. They are write-through and write-back [7].

1. Write-through: In this policy every write to the cache is reflected immediately to main memory.

2. Write-back: 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. In this paper write- back mechanism is used.

Operations performed on cache four operations are performed on cache [8]. They are:

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

Read hit

If processor wants to read a value from a location and if that location is found in cache then read hit is said to be occurred.

Read miss

If processor wants to read a value from a location and if that location is not found in cache then read miss occurs.

Write hit

If processor wants to write a value to a location and if that location is found in cache then write hit is said to be occurred.

Write miss

If processor wants to write a value to a location and if that location is not found in cache then write miss occurs. Now the processor writes that value in the main memory and caches that value.

CACHE Design

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 figure 3.2 .

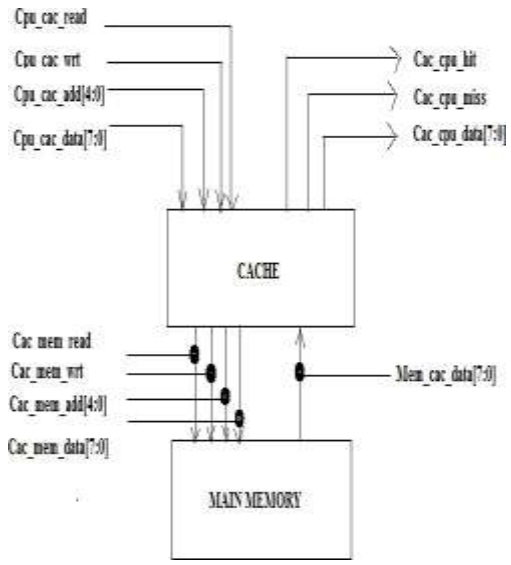

Figure 3.2: Architectural view

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

Simulation results for single cache

Simulation result for read hit

When reset (rst_1) is '0', the cache memory is loaded with the initial values. When rst_1 is '1', the cpu sends read signal (cpu_cac_read) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cache is "11000". The cpu_cac_add is divided into cpu_tag and cpu_index. The MSB 2bits indicate the cpu_tag and the LSB 3bits indicate the cpu_index. Therefore, cpu_tag is "11" and cpu_index is "000". In cache at '0' index the value present is 011100000100. The MSB bit indicates the dirty bit, 10^{th} bit indicates the valid bit, 9^{th} and 8^{th} bits are tag bits(cur_tag), 7 to 0 bits are the data bits. At 20ns the cpu sends the read signal. The cache compare the cpu_tag and cur_tag. As both are matched, the cache sends the hit (cac_cpu_hit) signal and data(cac_cpu_data) i.e.,00000100 to the cpu.

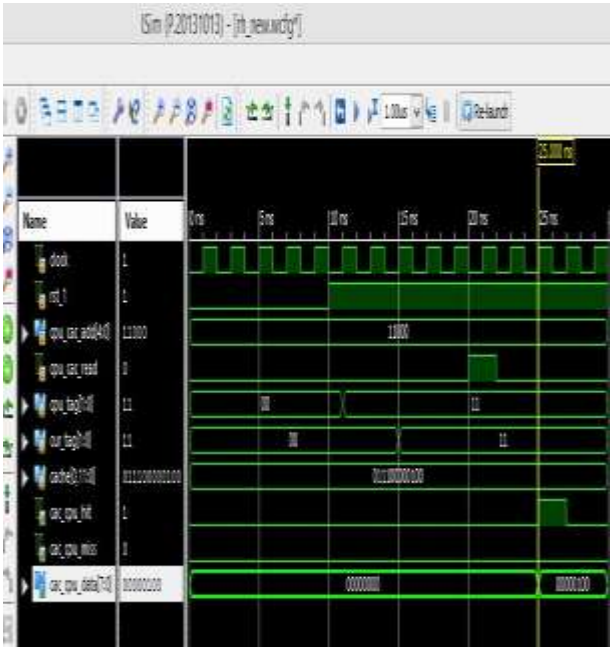

Figure 4.1.2: Cache read hit

Simulation result for read miss

When reset (rst_1) is '0', the cache memory is loaded with the initial values. When rst_1 is '1', the cpu sends read signal(cpu_cac_read) and address(cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cache is "01101". The cpu_cac_add is divided into cpu_tag and cpu_index. The MSB 2bits indicate the cpu_tag and the LSB 3bits indicate the cpu_index. Therefore, cpu_tag is "01" and cpu_index is "101". In cache at index 5, the value present is 011000001010. The MSB bit indicates the dirty bit, 10^{th} bit indicates the valid bit, 9^{th} and 8^{th} bits are tag bits(cur_tag), 7 to 0 bits are the data bits. At 20ns the cpu sends the read signal. The cache compares the cpu_tag and cur_tag. The cpu_tag is "01" and cur_tag is "10". There is a mismatch, therefore cache sends cac_cpu_miss signal to the CPU. The cache reads the data from memory and place the value in cache. Now the cache contains 010100001110. After placing the value in cache, the cache sends cac_cpu_hit signal and cac_cpu_data i.e., 00001110 to the cpu.

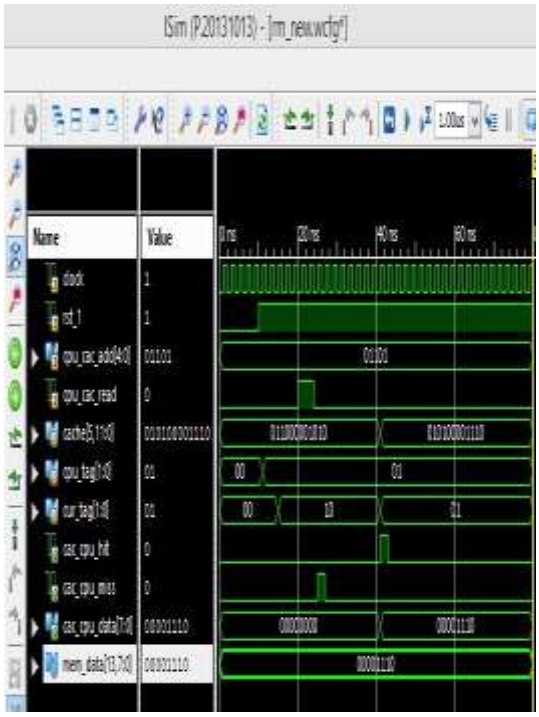

Figure 4.1.3: Cache read miss

Simulation results for write hit

When reset (rst_1) is '0', the cache memory is loaded with the initial values. When rst_1 is '1', the cpu sends write signal (cpu_cac_wrt), data (cpu_cac_data) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cache is "11000" and cpu_cac_data is "00001000". The cpu_cac_add is divided into cpu_tag and cpu_index. The MSB 2bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag is "11" and cpu_index is "000". In cache at '0' index the value present is 011100000100. The MSB bit indicates the dirty bit, 10^{th} bit indicates the valid bit, 9^{th} and 8^{th} bits are tag bits(cur_tag), 7 to 0 bits are the data bits. At 20ns the cpu sends the write signal and cpu_cac_data. The cache compares the cpu_tag and cur_tag. As both are matched, the cache sends the hit (cac_cpu_hit) signal and writes the new data into the cache. Now the cache contains 111100001000. The dirty bit is set to 1 in the cache entry to indicate that the data present in the cache is modified and different from that present in main memory.

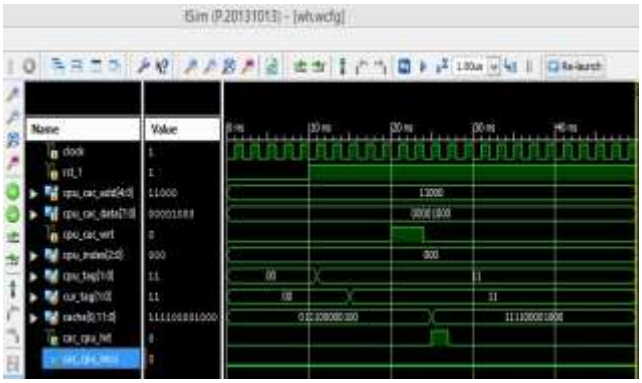

Figure 4.1.4: Cache write hit

Simulation results for write miss and read hit

When reset (rst_1) is '0', the cache memory is loaded with the initial values. When rst_1 is '1', the cpu sends write signal(cpu_cac_wrt),data (cpu_cac_data) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cache is "10000" and cpu_cac_data is "00001000". The cpu_cac_add is divided into cpu_tag and cpu_index. The MSB 2bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag is "10" and cpu_index is "000". In cache at '0' index the value present is 011100000100. The MSB bit indicates the dirty bit, 10^{th} bit indicates the valid bit, 9^{th} and 8^{th} bits are tag bits(cur_tag), 7 to 0 bits are the data bits. At 20ns the cpu sends the write signal and cpu_cac_data. The cache compares the cpu_tag and cur_tag. There is a mismatch, the cache sends the miss (cac_cpu_miss) signal and writes the new data into the cache. Now the cache contains

111000001000. The dirty bit is set to 1 in the cache entry to indicate that the data present in the cache is modified and different from that present in main memory. After 20ns the cpu sends read signal. As the tags matched, the cache sends cac_cpu_hit signal and cac_cpu_data 00001000 to the cpu.

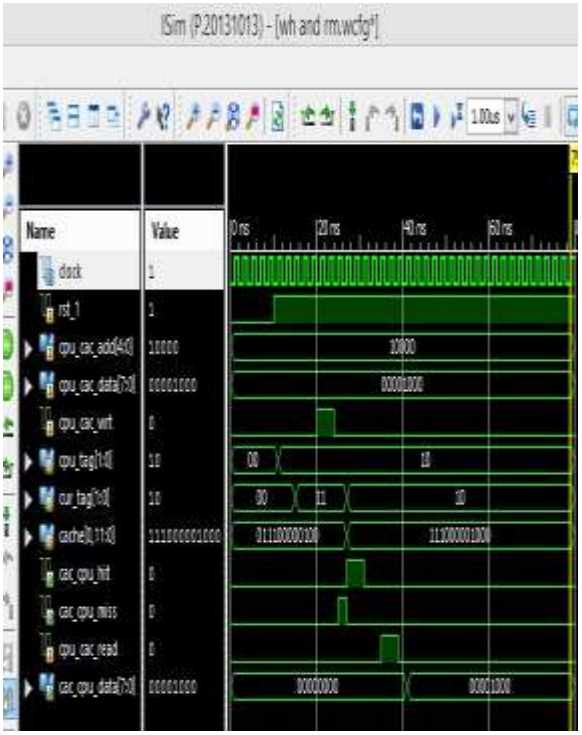

Figure 4.1.5: Cache write miss and read hit

Simulation result for write back

When reset (rst_1) is '0', the cache memory is loaded with the initial values. When rst_1 is '1', the cpu sends write signal(cpu_cac_wrt),data (cpu_cac_data) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cache is "10000" and cpu_cac_data is "00001000". The cpu_cac_add is divided into cpu_tag and cpu_index. The MSB 2bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag is "10" and cpu_index is "000". In cache at '0' index the value present is 011100000100. The MSB bit indicates the dirty bit, 10^{th} bit indicates the valid bit, 9^{th} and 8^{th} bits are tag bits(cur_tag), 7 to 0 bits are the data bits. At 20ns the cpu sends the write signal and cpu_cac_data. The cache compares the cpu_tag and cur_tag. There is a mismatch, the cache sends the miss (cac_cpu_miss) signal and writes the new data into the cache. Now the cache contains 111000001000. The dirty bit is set to 1 in the cache entry to indicate that the data present in the cache is modified and different from that present in main memory. After 12ns the cpu sends read signal and address. 11000. The tags are not matched so cac_cpu_miss signal is sent to the processor. The controller checks the dirty bit

before placing a new cache entry. The dirty bit is set to 1 so, the controller writes back the modified data to main memory and clears the dirty bit. Now new entry is placed and tags got matched so, cac_cpu_hit signal and cac_cpu_data is 00000100 is sent to the processor.

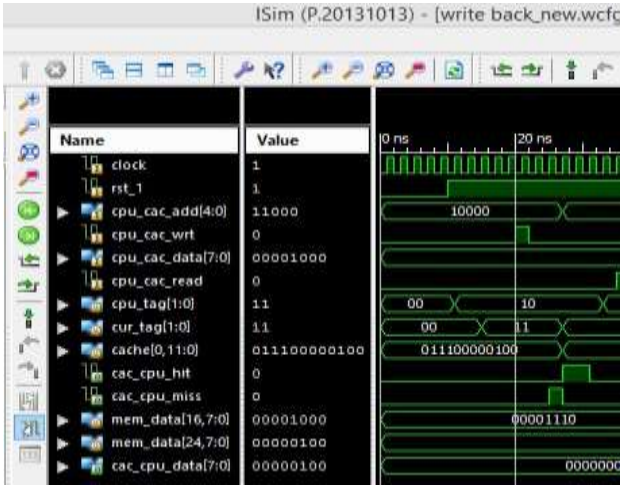

Figure 4.1.6: Cache write back mechanism

Simulation result for read hit and read miss

When reset (rst_1) is '0', the cache memory is loaded with the initial values. When rst_1 is '1', the cpu sends read signal(cpu_cac_read and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cache is "11000". The cpu_cac_add is divided into cpu_tag and cpu_index. The MSB 2bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag is "11" and cpu_index is "000". In cache at '0' index the value present is "011100000100". The MSB bit indicates the dirty bit, 10^{th} bit indicates the valid bit, 9^{th} and 8^{th} bits are tag bits(cur_tag), 7 to 0 bits are the data bits. At 20ns the cpu sends the write signal and cpu_cac_data. The cache compares the cpu_tag and cur_tag. There is a match ,therefore the cache sends the hit (cac_cpu_hit) signal and cac_cpu_data "00000100".After 10ns the cpu sends cpu_cac_read and address 01001 to the cache. The cpu_tag is "01" and cur_tag is "10". There is a mismatch in tags so, cac_cpu_miss signal is sent to the processor. The data is now read from memory and placed in cache. Now the cache index 1 contains "010110001000". Now tags are matched, so cac_cpu_hit signal and cac_cpu_data "10001000" is sent to the cpu.

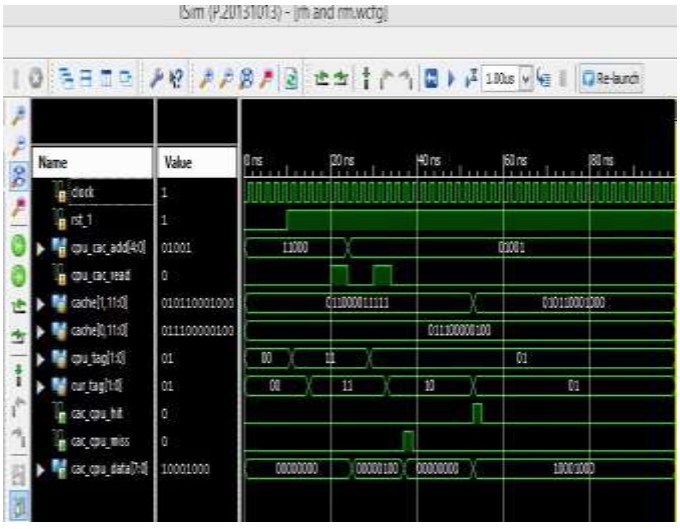

Figure 4.1.7: Cache read hit and read miss

Results

Simulation results

Simulation result for local cache read hit

When reset (rst_1) is '0', the cache memories are loaded with the initial values. When rst_1 is '1', the cpu sends read signal(cpu_cac_read) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cacheA is "00111". The cpu_cac_add is divided into cpu_tag_A and cpu_index_A. The MSB 2 bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag_A is "00" and cpu_index_A is "111". In cacheA at index 7, the value present is "01010000010100". The MSB 2bits indicate the states,the dirty bit, 10^{th} bit indicates the dirty bit, 9^{th} bit indicates the valid bit, and 8^{th} and 7^{th} bits indicate the tag bits(cur_tag), 8 to 0 bits are the data bits. At 12ns the cpu sends the read signal.The cache compares the cpu_tag_A and cur_tag_A. There is a match, therefore the cache sends the hit (cac_cpu_hit_A) signal and cac_cpu_data_A "00010100" to the cpu.

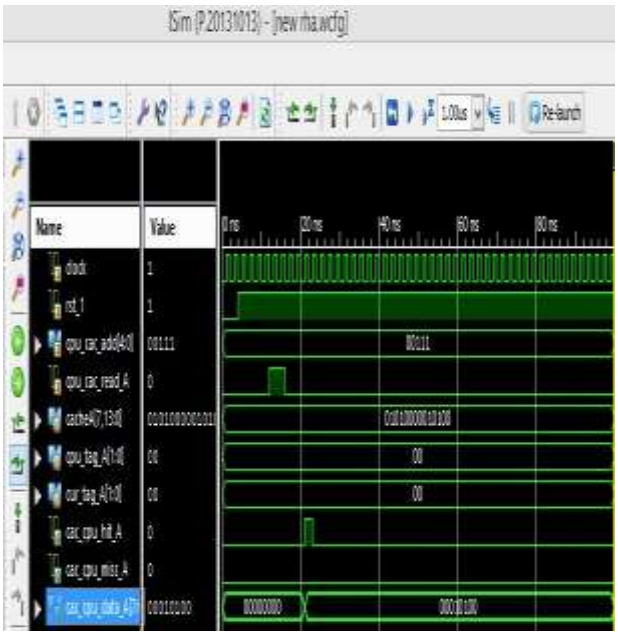

Figure 5.1: Local cache read hit

Simulation result for local cache read miss

When reset (rst_1) is '0', the cache memories are loaded with the initial values. When rst_1 is '1', the cpu sends read signal(cpu_cac_read) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cacheA is "10111". The cpu_cac_add is divided into cpu_tag_A and cpu_index_A. The MSB 2 bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag_A is "10" and cpu_index_A is "111". In cacheA at index 7, the value present is "01010000010100". The MSB 2bits indicate the states,the dirty bit, 10^{th} bit indicates the dirty bit, 9^{th} bit indicates the valid bit, and 8^{th} and 7^{th} bits indicate the tag bits(cur_tag), 8 to 0 bits are the data bits. At12ns the cpu sends the read signal.The cache compares the cpu_tag_A and cur_tag_A. There is a mismatch, therefore the cache sends the miss(cac_cpu_miss_A) signal to the cpu.Now the controller checks whether the address is present in cacheB. The cpu_tag_B and cur_tag_B are same so, the data is placed in cacheA and both the states are changed to Shared state. The cac_cpu_hit_A and cac_cpu_data_A "00001001" are sent to the cpu.

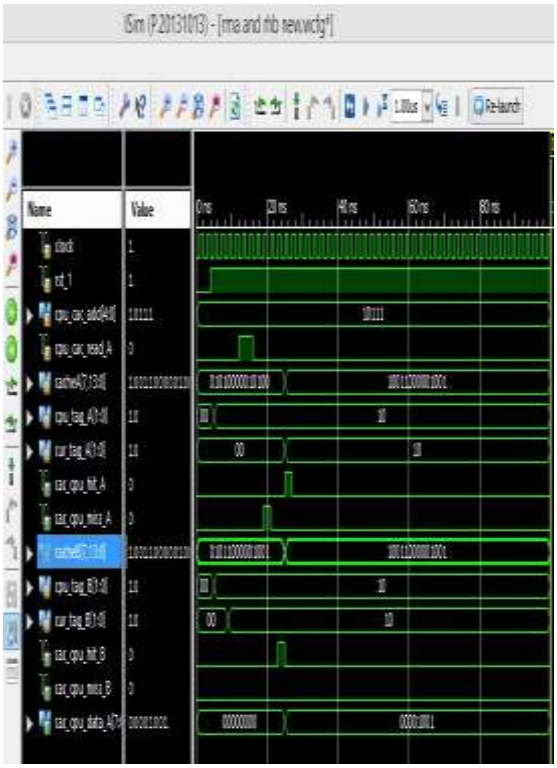

Figure 5.2: RMA, RHB

Simulation result for local cache read miss

When reset (rst_1) is '0', the cache memories are loaded with the initial values. When rst_1 is '1', the cpu sends read signal(cpu_cac_read) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cacheA is "10110". The cpu_cac_add is divided into cpu_tag_A and cpu_index_A. The MSB 2 bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag_A is "10" and cpu_index_A is "110". In cacheA at index 6, the value present is "01010100000010". The MSB 2bits indicate the states,the dirty bit, 10^{th} bit indicates the dirty bit, 9^{th} bit indicates the valid bit, and 8^{th} and 7^{th} bits indicate the tag bits(cur_tag), 8 to 0 bits are the data bits. At 12ns the cpu sends the read signal.The cache compares the cpu_tag_A and cur_tag_A. There is a mismatch, therefore the cache sends the miss(cac_cpu_miss_A) signal to the cpu.Now the controller checks whether the address is present in cacheB. The cpu_tag_B and cur_tag_B are not same so, the controller checks for the address in cacheC. The cpu_tag_C and cur_tag_C are matched. Now the controller places this value into cacheA and both the cache states are changed to shared. The cac_cpu_hit_A and cac_cpu_data_A "00011110" are sent to the processor.

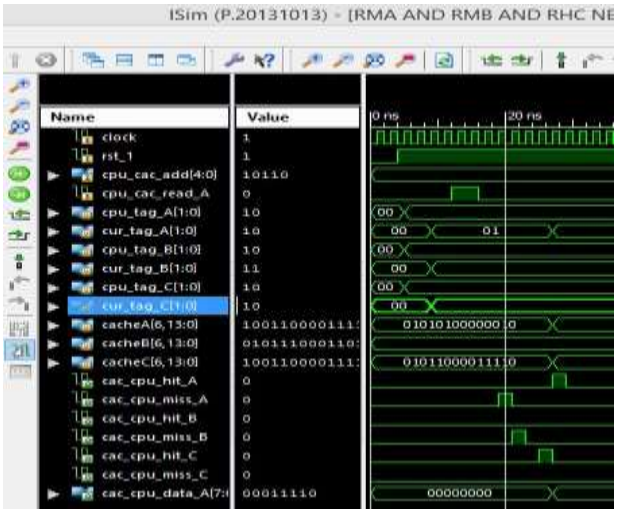

Figure 5.3: RMA, RMB, RHC

Simulation result for local cache read miss

When reset (rst_1) is '0', the cache memories are loaded with the initial values. When rst_1 is '1', the cpu sends read signal(cpu_cac_read) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cacheA is "01111". The cpu_cac_add is divided into cpu_tag_A and cpu_index_A. The MSB 2 bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag_A is "01" and cpu_index_A is "111". In cacheA at index 7, the value present is "01010000010100". The MSB 2bits indicate the states,the dirty bit, 10^{th} bit indicates the dirty bit, 9^{th} bit indicates the valid bit, and 8^{th} and 7^{th} bits indicate the tag bits(cur_tag), 8 to 0 bits are the data bits. At 12ns the cpu sends the read signal.The cache compares the cpu_tag_A and cur_tag_A. There is a mismatch, therefore the cache sends the miss(cac_cpu_miss_A) signal to the cpu.Now the controller checks whether the address is present in cacheB. The cpu_tag_B and cur_tag_B are not same so, the controller checks for the address in cacheC. The cpu_tag_C and cur_tag_C are not matched. Now the controller searches for the location in main memory. After reading the value from memory,the controller puts this value in cacheA and the state is changed to exclusive.

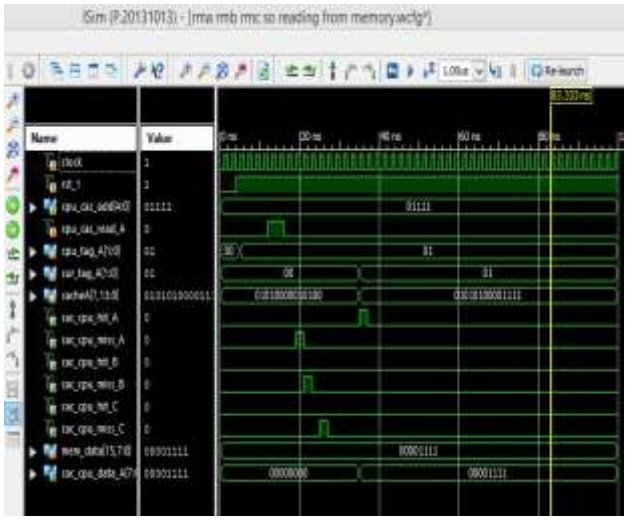

Figure 5.4: RMA, RMB, RMC, Reading from memory

Simulation result for local cache write hit

When reset (rst_1) is '0', the cache memories are loaded with the initial values. When rst_1 is '1', the cpu sends write signal(cpu_cac_write), data(cpu_cac_data_A) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cacheA is "11000". The cpu_cac_add is divided into cpu_tag_A and cpu_index_A. The MSB 2 bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag_A is "11" and cpu_index_A is "000". In cacheA at index 0, the value present is "01011100000100". The MSB 2bits indicate the states,the dirty bit, 10^{th} bit indicates the dirty bit, 9^{th} bit indicates the valid bit, and 8^{th} and 7^{th} bits indicate the tag bits(cur_tag), 8 to 0 bits are the data bits. At 12ns the cpu sends the write signal and data.The cache compares the cpu_tag_A and cur_tag_A. There is a match, therefore the cache sends the hit(cac_cpu_hit_A) signal to the cpu. The dirty bit is set to'1' and state is changed from exclusive to modified. After writing the data into cache, now at index '0' the value present is "00111100000001".

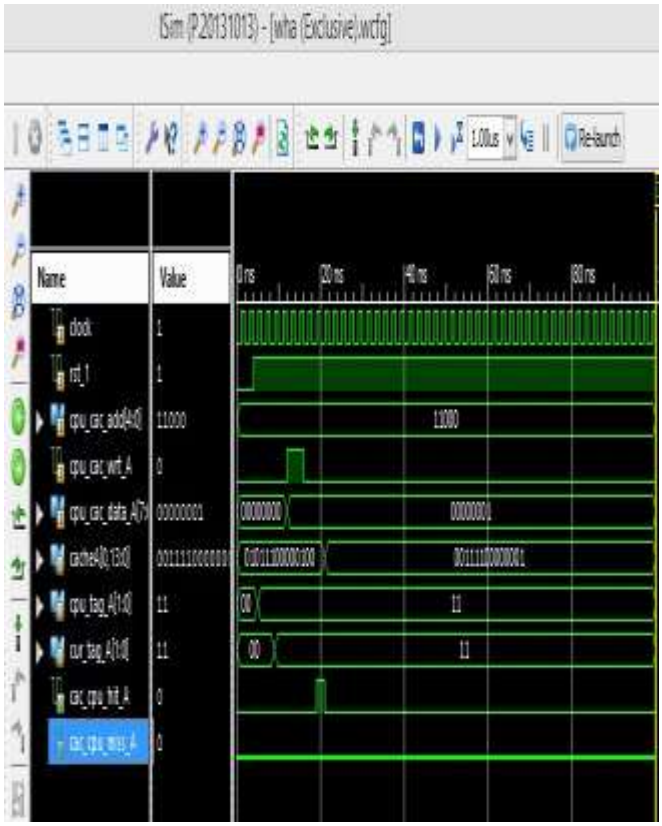

Figure 5.5: WHA (Exclusive state)

Simulation result for local cache write hit:

When reset (rst_1) is '0', the cache memories are loaded with the initial values. When rst_1 is '1', the cpu sends write signal(cpu_cac_write), data(cpu_cac_data_A) and address (cpu_cac_add) to the cache. The cpu_cac_add sent by the cpu to cacheA is "11100" and cpu_cac_data_A is "00000001".The cpu_cac_add is divided into cpu_tag_A and cpu_index_A. The MSB 2 bits indicate the cpu_tag and the LSB 3 bits indicate the cpu_index. Therefore, cpu_tag_A is "11" and cpu_index_A is "100". In cacheA at index 4, the value present is "10011100000101". The MSB 2bits indicate the states,the dirty bit, 10^{th} bit indicates the dirty bit, 9^{th} bit indicates the valid bit, and 8^{th} and 7^{th} bits indicate the tag bits(cur_tag), 8 to 0 bits are the data bits. At 12ns the cpu sends the write signal and data.The cache compares the cpu_tag_A and cur_tag_A. There is a match, therefore the cache sends the hit(cac_cpu_hit_A) signal to the cpu. In cacheA the state is shared. As we are performing write operation to a shared location, the controller checks for the same location in other caches. CacheB also

contains the same address, therefore controller changes the CacheB state to invalid and cacheA state changed from shared to modified.Now the cacheA contains “00111100000001”.

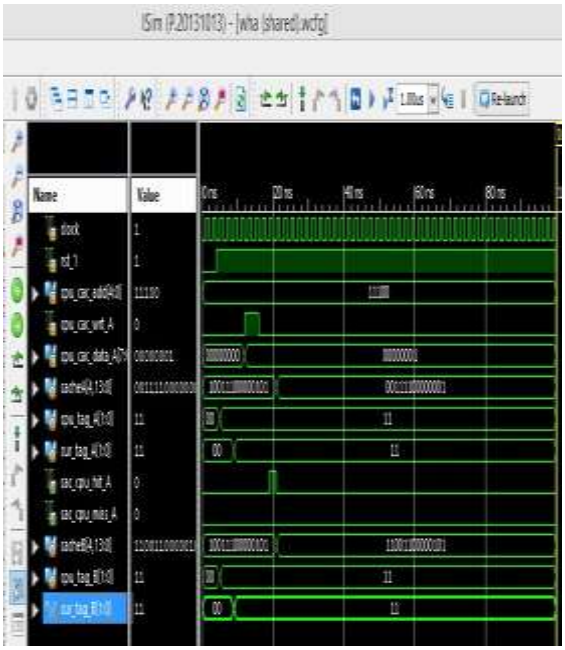

Figure 5.6: WHA (Shared state)

Simulation result of cache coherence in multiprocessor systems

The processor A sends address and read signal to cacheA. The tags are matched so, it sends a hit signal(cac_cpu_hit_A) to the cpuA. After 20ns cpuB sends address and read signal to cacheB. The tags are matched so, it sends a hit signal(cac_cpu_hit_B) to the cpuB. The processor sends address and write signal to cacheA after 10ns. The tags are matched. The cacheA state changed from shared to modified and now the controller checks for other caches having the same address. The cacheB also holds a copy of data for the same address, so the controller changes its state from shared to invalid and the new data is written back to main memory. After 10ns the processor sends read signal, the tags are not matched so it reads the data from memory. As the address is present in other caches, the state of cacheC changed to shared. So, now cacheC reads the modified data.

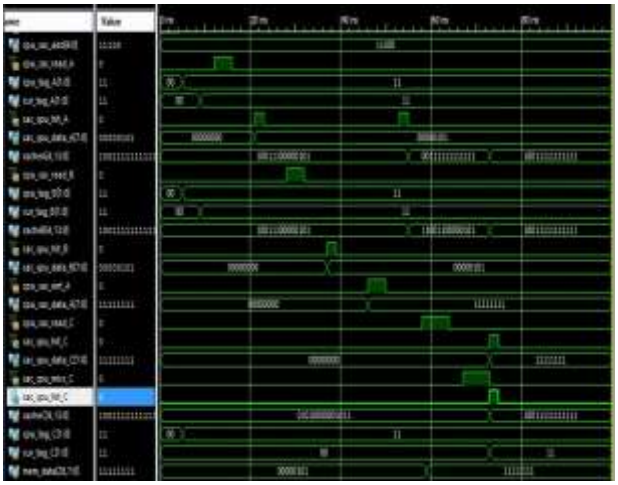

Figure 5.7: Cache coherence in multiprocessor system

CONCLUSION

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. Therefore, for every shared location we need to write back the modified data into main memory. Hence, the numbers of clock cycles required are more. Therefore, a better technique can be developed to reduce the number of write backs which in turn reduces the number of clock cycles.

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) 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.
9) 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.
10) D.A.Patterison, J.I.Hennessy.1998.Computer organization and design.4th edition.
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.

.