Design and Implementation of Cache Coherence Protocol for High-Speed Multiprocessor System

Daman Preet Kaur
Academic and Consultancy Services Division
Centre for Development of Advanced Computing
Mohali, India
dpkaur.165@gmail.com

V. Sulochana
Academic and Consultancy Services Division
Centre for Development of Advanced Computing
Mohali, India
vemus@cdac.in

***Abstract*—To maintain data consistency between the cache memories in centralized and distributed shared-memory multiprocessor system, particular protocols are used known as cache coherence protocols. The performance of a multi-core computer system is strongly influenced by the type of cache coherence protocol used. In this paper, the snoopy bus cache coherence protocols using 3-state, 4-state and 5-state are designed and implemented using the write-invalidate approach in a shared memory dual processor system. The simulation results show that the MOESI protocol reduces the load misses by 48%, memory latency by 35%, power consumption by 18%, increases the gate count by 34% and improves the execution time by 22%. Thus, the overall performance of the MOESI is better than the MESI and MSI cache coherence protocols in a shared memory dual processor system.**

Keywords—*Cache coherence protocols, MSI, MESI, MOESI, Shared memory multiprocessors*

I. INTRODUCTION

In today's computer architectures, shared memory multi-core processors are becoming dominant. But, the processor takes too many cycles to access the main memory in multi-core architectures because of the speed difference between the main memory and processor [1]. So, for tuning up the performance of multi-core architectures large multilevel caches are used in the system that helps to reduce the memory access time of the processor. These caches also help to reduce demand of memory bandwidth of a processor [2]. Due to the reduction in the main memory bandwidth demand of a single processor, multiple processors are able to share the same memory. Such symmetric shared memory multiprocessors are extremely cost-effective that can perform caching of both shared and private data as shown in Fig. 1. The private data is cached only by the single processor so the program behavior is similar to that in the uniprocessor while shared data is replicated in the multiple caches [3].

Along with the reduction in memory bandwidth and access latency, the reduction in the contention is also provided by the replication of data. The contention takes place when the multiple processors read the shared data simultaneously [4]. However, in spite of all these advantages, the hindrance called cache coherence problem was raised due to caching of shared data.

The cache coherence problem may occur due to data inconsistency between the caches and the shared memory during sharing of modified data, process migration and I/O operation [5]. When the cache memories of multiple processors maintain local copies of the same data object taken from the main memory and even if any one of the caches modifies the value of same data object then that will result in globally inconsistent view of the shared data between caches and shared memory [6]. This problem is called cache coherency. This problem may occur when multiple processors operate independently and asynchronously. To eliminate this problem in the memory system defined protocols are used [7]. These protocols are known as cache coherence protocols. Different kinds of cache coherence protocols can be implemented according to the different multiprocessors based systems [8].

In this paper, the classification of cache coherence protocols and snoopy bus protocols are described in section II. The design and implementation of MSI, MESI, and MOESI protocols using dual-core architecture are shown in section III and finally in section IV conclusion and future scope is presented.

II. CACHE COHERENCE PROTOCOLS

The specific set of rules executed for maintaining coherency in multi-core processors are known as cache coherence protocols. The numbers of cache coherence protocols are available for maintaining coherency in the multiprocessor system. Coherency helps in managing the read and write operations in the shared cache to have the consistent view among all the processors. The fundamental action of a cache coherence protocol is to find the state of shared data blocks. Cache coherence protocols are classified into two categories that are directory-based protocols and snoopy bus protocols [8] [9].

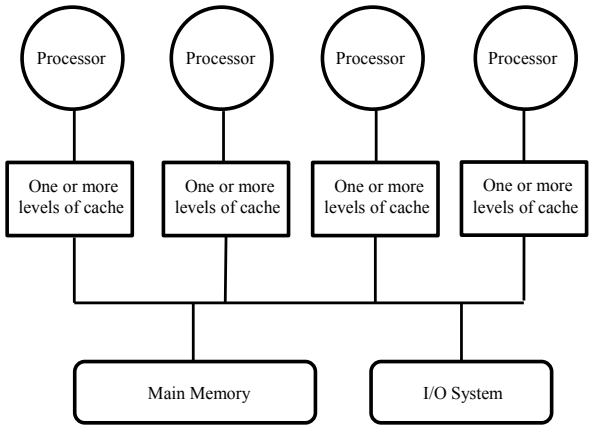


Fig. 1. The basic structure of a shared memory multiprocessor system

978-1-5386-6625-8/18/$31.00 ©2018 IEEE

Directory-based protocols are used in the multistage network in which hundreds of processors are connected by network [10]. These protocols have one location called the central directory which records the state and presence information for each cache block. They support distributed shared-memory architectures.

Snoopy bus protocols are used by the bus-based memory system. Unlike directory protocols, there is not any specified region in snoopy bus protocols that stores the information of data items that are being shared but instead of that these protocols make use of the global bus [10] [11]. The bus is a very convenient source for maintaining cache coherency by allowing all multiple processors to observe ongoing memory transactions [12]. In the bus-based memory system, each cache has its own cache controller that continuously monitors or snoops on the global bus to detect the requested data block copy on the bus [13]. These protocols support centralized shared-memory architectures.

The two main approaches used to implement snoopy bus protocols are write-update and write-invalidate. In write invalidate approach if any one local cache updates its value then the data copies of all other caches are invalidated. In write update approach, if the data item is written in any one cache then it will get updated in all caches. There is more bandwidth consumption in write-invalidate approach because of the broadcast of the write operation in all caches [14] [15]. That is why the use of the write invalidate approach is always preferred in many bus protocols.

In this work, MSI, MESI, and MOESI snooping protocols explained in detail below are designed and implemented in a dual-core system consist of two caches and the main memory. The main cache events and actions that are triggered by the main memory and processor in protocols are explained in Table I. According to all these cache actions, the transitions between the different states of the cache take place [16] [17].

A. MSI (Modified-Shared-Invalid) Protocol

MSI protocol is called classic invalidate-based protocol. It is composed of three states. The state diagram for the MSI protocol is shown in Fig.2. The cache is in a modified state when the data block of the cache is modified and is different from the value of the data block that is in the main memory

TABLE I. CACHE EVENTS AND THEIR ACTIONS

Cache Events	Actions	
	Source	***Function***
Write hit (WH)	Processor	Write data in the cache
Read hit (RH)	Processor	Read data from the cache
Snoop hit on write (SHW)	Bus	Hit from another cache during write
Snoop hit on read (SHR)	Bus	Hit from another cache during read
Read Miss Exclusive (RME)	Bus	Request data from cache or memory
Read Miss Shared (RMS)	Bus	Request data from the other cache

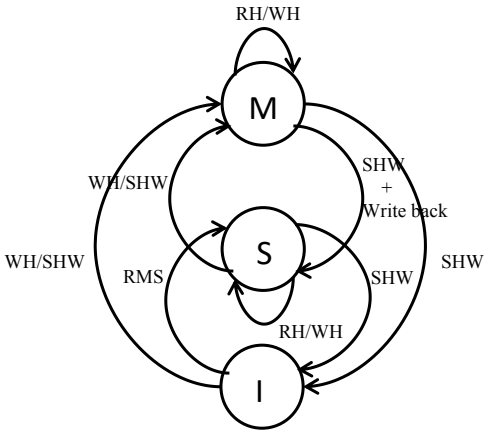


Fig. 2. State transition diagram of MSI Protocol

and other caches. This is the only cache whose data copy is valid [18]. The cache is in a shared state when the data block is shared among all the caches taken from the main memory. The cache is in the invalid state when the data block of the cache is of no use.

B. MESI (Modified-Exclusive-Shared-Invalid) Protocol

The most commonly used protocol that supports write-back cache is the MESI protocol. This is an extension of the MSI protocol in which the fourth state is added known as the *Exclusive* state that helps to reduce the number of bus transitions from an invalid state to a modified state [19]. The state transition for the MESI protocol is shown in Fig.3. When only one cache has a valid data block then that cache is known to be in the exclusive state and that valid data block is also updated in the main memory but not in the other caches. The use of the exclusive state is that without any need for snooping from other caches, local cache can modify itself independently.

C. MOESI (Modified-Owned-Exclusive-Shared-Invalid) Protocol

MOESI protocol is an improved version of the MESI

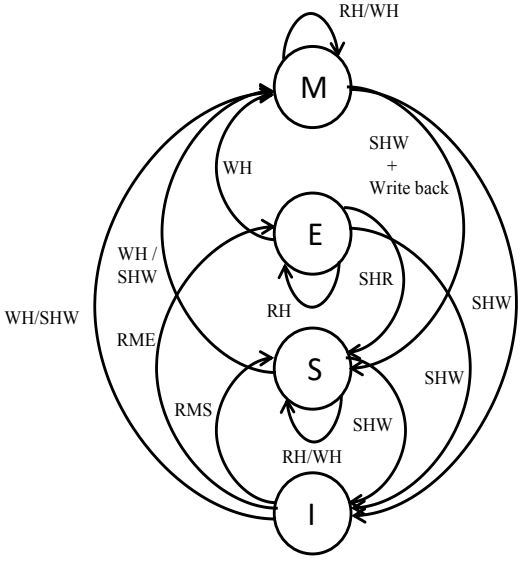


Fig. 3. State transition diagram of MESI Protocol

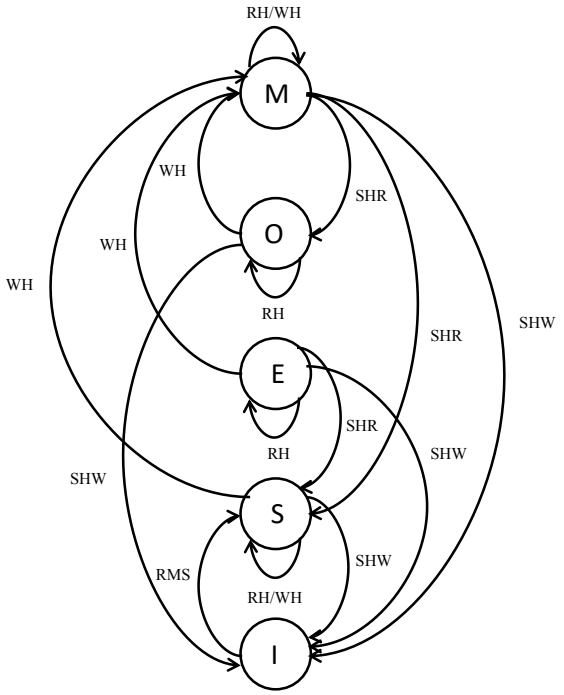


Fig. 4. State transition diagram of MOESIC Protocol

in which the fifth state is added called the owned state that helps to reduce the number of memory accesses [20]. The state diagram of the MOESI protocol is shown in Fig.4. In the multiprocessor system, the owned state cache containing the valid data block is only one and that cache can copy the valid data block to other caches directly in spite of sharing with the main memory to minimize access to the main memory. Thus, the owned state represents both the modified state and shared state. The dirty values can also be shared with other caches because the data in the main memory is not updated [21]. It means that owned state cache has a valid data block and other caches and the main memory may have a valid data block too at the same time. This is another advantage of this protocol.

III. Results

A. Design of Protocols

The state machines of write-invalidate snooping cache coherence protocols viz. MSI, MESI, and MOESI have been first designed using write-back cache block in Verilog HDL and simulated using ISim simulator. The different input signals like write hit (WH), snoop hit on write (SHW), read hit (RH) and, snoop hit on read (SHR) are applied in the cache to run the different protocols state wise. Fig.5, Fig.6, and Fig.8 show the successful state wise transition of MSI, MESI, and MOESI protocols in the simulation waveform. The different states of the two caches are represented by the three-bit signals named as SectorInCacheA and SectorInCacheB respectively.

The designed MSI, MESI and MOESI protocols are implemented in a dual processor system. The general design of shared memory dual processor system used to verify the functionality of snooping protocols consists of processor unit (PU), memory mapping unit (MMU), cache, memory bus controller (MBC) and memory as shown in Fig.8. All

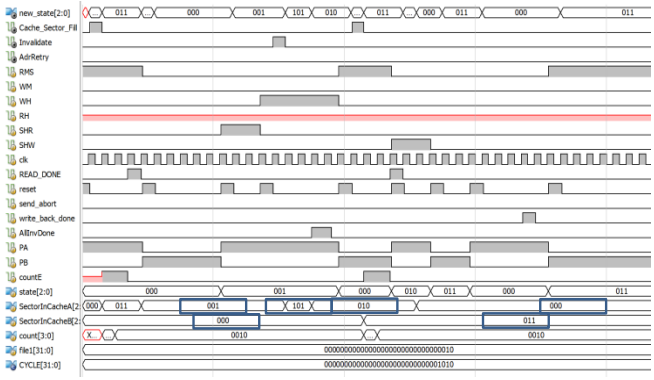


Fig. 5. State transitions of MSI Protocol

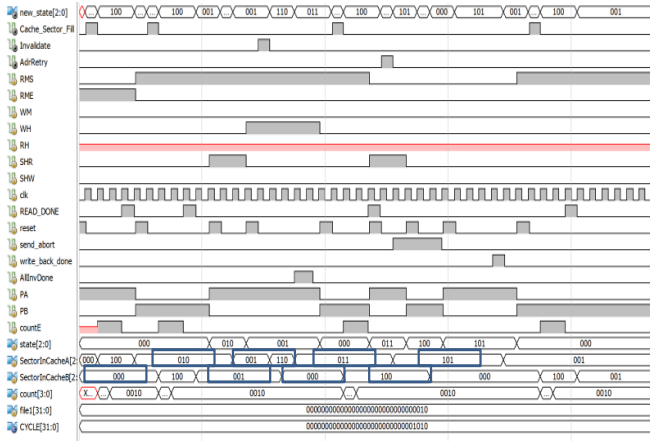


Fig. 6. State transitions of MESI Protocol

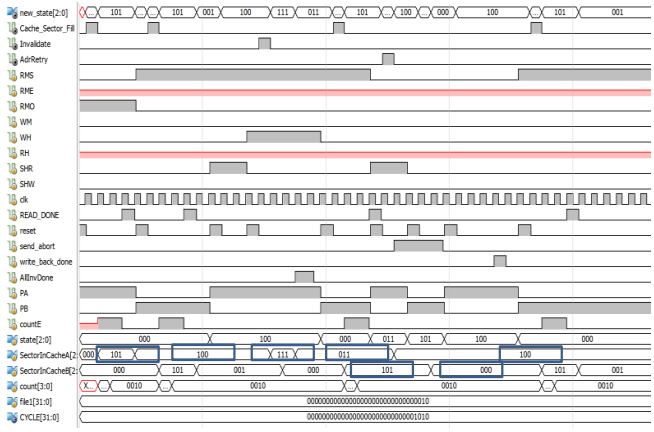


Fig. 7. State transitions of MOESI Protocol

modules of the dual-core system are simulated in ModelSim PE Student Edition. There are two caches – Cache A and Cache B in the design. The caches are two-way associative.

Each cache has 8 entries of 11 bits that includes data, tag, update, dirty and valid bits. The two processor unit Processor A and Processor B are used to perform write actions and read actions in the caches. The shared memory used in the design has 32 entries and each entry is of 10 bits as shown in Fig.9.

Memory mapping unit is used to convert virtual address to physical address by cutting the two most significant bits of the virtual address. Memory bus controller is used to make the synchronization between different modules that access the shared bus at the same time. The simulations of cache

and memory are carried out with the help of the memory mapping unit and memory bus controller. The simulation results of read and write operations in the cache are shown in Fig.10 and the memory read and writes operations are shown in Fig.11.

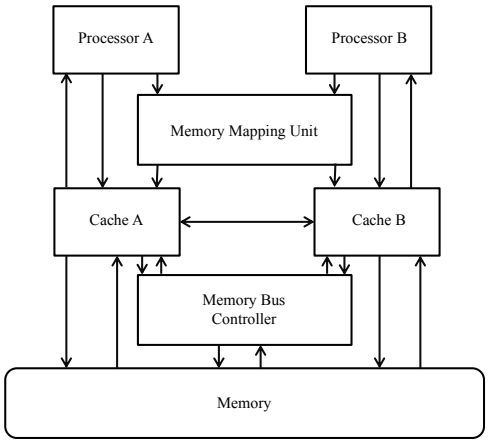


Fig. 8. General design of shared memory dual processor system

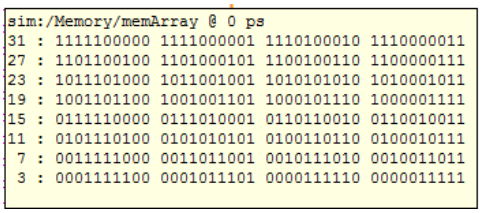


Fig. 9. Memory consists of 32 entries at reset condition

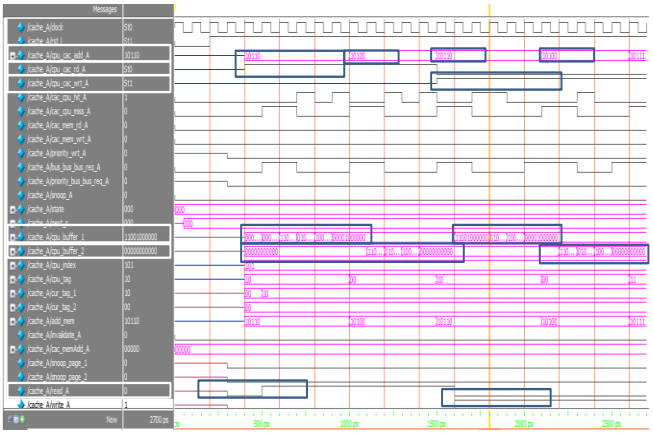


Fig. 10. Simulation showing read and write operations of cache A

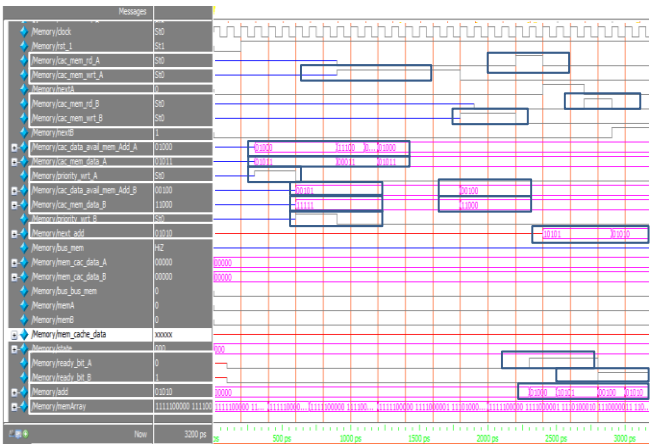


Fig. 11. Simulation showing read and write operations of the memory

B. *Implementation of Protocols*

All the designed modules of shared memory dual processor system are synthesized and implemented in Xilinx ISE Design Suite after successful simulation of each part of shared memory dual processor system. The register transfer level schematic of each part of the shared memory dual processor system is created that shows the various input and the output ports of each module. Internally it contains the connections among the different logics and the registers used to design the modules. The RTL schematics of cache A, cache B, memory, memory mapping unit and memory bus controller are shown in Fig.12.

The three different test benches are designed to verify the functionality of the MSI, MESI and MOESI snoopy bus protocols in the designed environment of shared memory dual core system. The designed environment is created as the top module by calling the instance of each module of the dual processor system. The required different input commands and cache requests are given in the test bench at the proper time intervals to run the transitions between the different states of the cache.

The write invalidate approach of a dual-core shared memory system is verified in test benches by changing the data of cache A that automatically invalidates the data of cache B according to the logic set up in the caches and shared memory during designing. The write-back approach of each cache block is also verified in the test bench by changing the data of each cache at the regular intervals but the same data is not updated in the main memory. Thus, both the write-invalidate approach and write back cache blocks helps out for improving the execution time of different protocols in the designed environment.

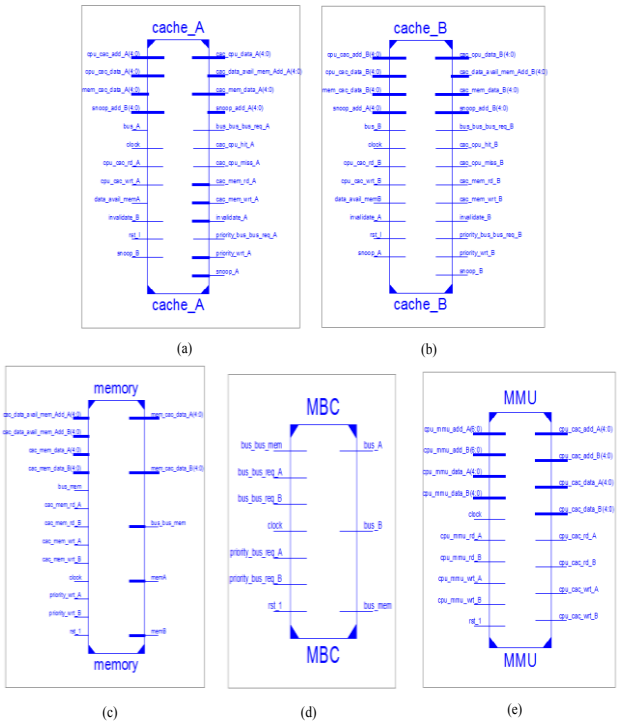
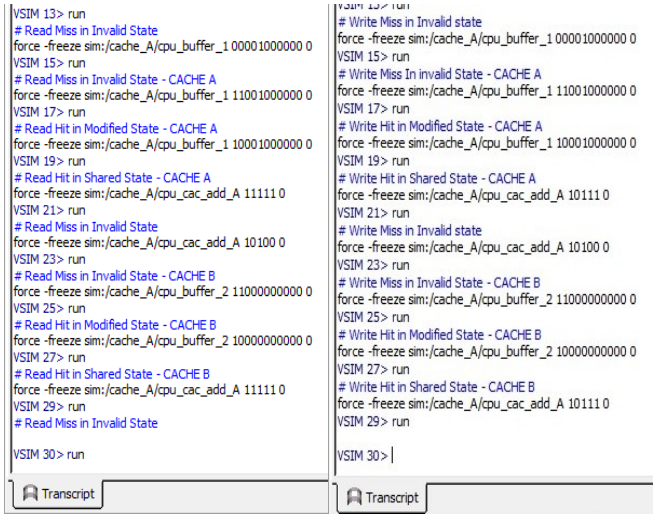


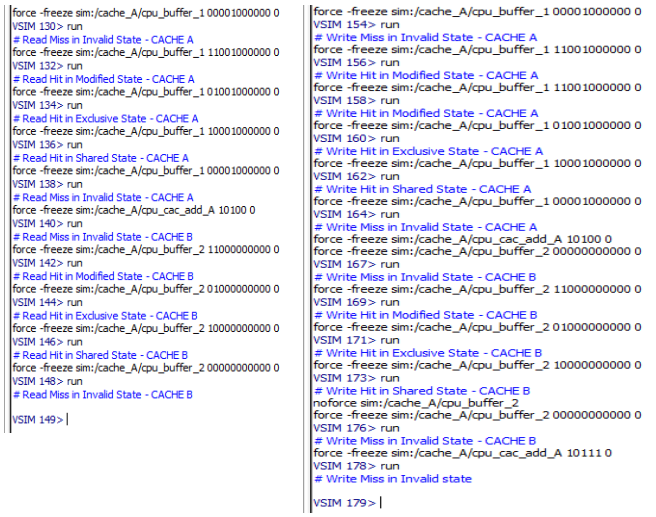
Fig. 12. RTL schematics of (a) cache A, (b) cache B, (c) memory, (d) memory bus controller (MBC), (e) memory mapping unit (MMU)

The different cache requests that are run within the system like read hit, write hit, read miss and write miss and

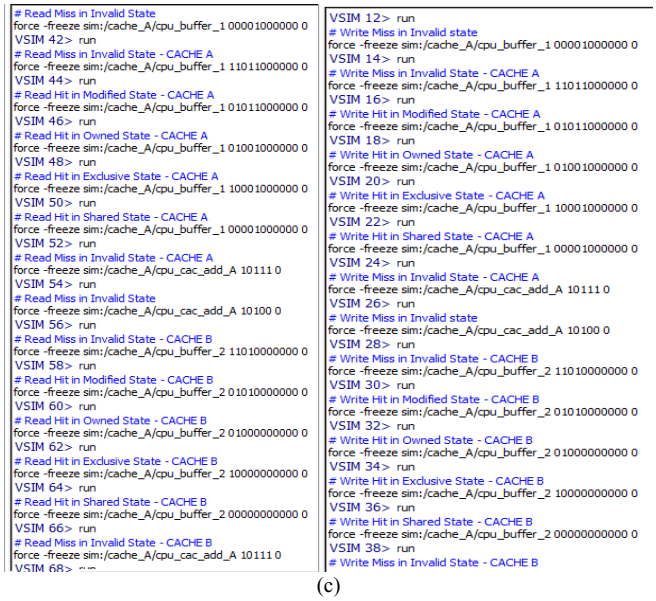
the different state transitions like modified, shared, invalid, exclusive and owned that takes place according to the cache action of MSI, MESI and MOESI protocols are displayed in Fig. 13. as the transcript to show the implementation of protocols in shared memory dual processor system.



(a)



(b)



(c)

Fig. 13. Transcripts showing different read and write actions and different state transitions of cache A and cache B blocks running in shared memory dual processor system of (a) MSI protocol, (b) MESI protocol and (c) MOESI protocol

C. *Comparison of Protocols*

The comparison section examines the performance of MSI, MESI and MOESI protocol in terms of load misses, memory latency, power consumption, gate count and execution time on the basis of the observation of the simulation and implementation results of MSI, MESI, and MOESI snooping protocols in shared memory dual processor system.

The comparison results as in Table II show that the MOESI protocol reduces the load misses by 48% and memory latency by 35% as the total number of main memory accesses are reduced by the owned state of MOESI protocol. The power consumption is also reduced by 18% and overall execution time is improved by 22% because of the use of both the write-invalidate approach and write back cache blocks in the MOESI protocol. However, the gate count is increased by 34% because the numbers of states are more in the MOESI protocol. Therefore, results demonstrate the superiority of the MOESI protocol in comparison with the MSI and MESI protocol because it has low power consumption and less delay.

TABLE II. COMPARISON OF MSI, MESI AND MOESI PROTOCOLS

Protocol	Comparison Parameters				
	Load Misses	***Gate Count***	***Memory Latency***	***Power consump-tion***	***Execution Time***
MSI	128	6681	120 ns	61 %	372 ns
MESI	105	8264	93 ns	52 %	290 ns
MOESI	61	10192	42 ns	43 %	82 ns

IV. CONCLUSION

The performance of multi-core computer systems is strongly influenced by the type of cache coherence protocol used. For the different kinds of shared memory multiprocessor architectures, the suitable cache coherence protocol should be used because it has a direct impact on memory latency, power consumption and execution time of the system.

The results provided in this paper are according to shared memory dual-core system. The performance of dual processor system is observed by using write-invalidate MSI, MESI and MOESI snoopy bus protocols using write- back cache blocks. Based on the simulation and implementation results it is observed that MOESI protocol which is an add-on of MESI and MSI protocol minimizes the memory latency, power consumption and execution time. However, the area utilization of MOESI protocol is more as compare to MESI and MSI protocol. Therefore it is concluded that the overall performance of MOESI protocol is better than MESI and MSI protocols in a shared memory dual processor system.

Further, this work can be expanded by implementing

MSI, MESI, and MOESI snooping protocols on the real-time system like FPGA boards. The future scope of this work is to assess the performance of the shared memory system by increasing the number of processors or by implementing other snoopy bus protocols like dragon and firefly write update based protocols.

REFERENCES

[1] Hennessy, J.L., Patterson, and D.A, "Multiprocessor and Thread-Level Parallelism," *in Computer Architecture: A Quantitative Approach,* 4th ed., San Francisco,California, USA: M K Publishers, 2007, pp. 195-286

[2] Hesham Altwaijry and Diyab S.Alzahrani, "Improved-MOESI Cache Coherence Protocol," Arab J Sci Eng. Springer-Verlag, Berlin Heidelberg New York, vol.39., pp. 2739-2748, 2014.

[3] Kyueun Yi, Won W. Ro, and Jean-Luc Gaudiot., "Importance of Coherence Protocols with Network Applications on Multicore Processors," IEEE Transactions on Computers, IEEE Press, New York vol.62., pp. 6-15, 2013.

[4] Amit D. Joshi, S.V.; B.Sham, and N.Rama, "Performance Analysis of Cache Coherence Protocols for Multi-core Architectures" AICTC, ACM, New York, 2016.

[5] Somdip Dey, and Mamatha S. Nair "Design and Implementation of a Simple Cache Simulator in Java to Investigate MESI and MOESI Coherency Protocols," International Journal of Computer Applications, IEEE Press, New York, vol.87, pp. 0975 – 8887, 2014.

[6] Alberto Ros, Manuel E.Acacio, and Jose M.Garcıa, "A Direct Coherence Protocol for Many-Core Chip Multiprocessors" IEEE Transactions On Parallel And Distributed Systems, IEEE Press, New York, vol.21, pp. 1779-1792, 2010.

[7] Daniel J Sorin, Mark D Hill, and David A Wood, "A primer on memory consistency and cache coherence," in *Synthesis Lectures on Computer Architecture:* Morgan & Claypool Publishers, 2011,pp. 1 - 214

[8] Kai Hwang , "Multiprocessors and Multicomputers," *in Advanced Computer Architecture,* 2nd ed., New Delhi, India: Tata McGraw-Hill Edu. Publishers, 2011, pp. 281-340

[9] Suh, S., "Integration and Evaluation of Cache Coherence Protocols for Multiprocessor SOCS.Ph.D," Thesis Georgia Tech, USA, 2006.

[10] Lluc Al, Lluís V., Marc G., Xav. M., and Eduard Ayg, "Hardware–Software Coherence Protocol for the Coexistence of Caches and Local Memories," IEEE Transactions on Computers, IEEE Press, New York, vol.64, pp. 152-165, 2015.

[11] Hackenberg, D.; Molka, D., Nagel, and W.E "Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems In," Micro-42 Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, New York 2009.

[12] Felix Garcia-C., Jesus Carr, Alejandro Cal., Jose M. Perez, and Jose D. G "An Adaptive Cache Coherence Protocol Specification for Parallel Input/output Systems" IEEE Transactions On Parallel And Distributed Systems, IEEE Press, New York, vol.15, pp. 533-545, 2004

[13] Diana Keen, Mark Oskin, Justin Hensley, and Frederic T. Chong, "Cache Coherence in Intelligent Memory Systems," IEEE Transactions on Computers, IEEE Press, New York, vol.52, pp. 960-966, 2003.

[14] Sabela Ramos, and Torsten Hoefler, "Cache Line Aware Algorithm Design for Cache-Coherent Architectures," IEEE Transactions On Parallel And Distributed Systems, vol.27, pp.2824-2837, October 2016.

[15] Daehoon Kim, Chang Hyun Park, Hwanju Kim, and Jaehyuk Huh, " Virtual Snooping Coherence for Multi-Core Virtualized Systems," IEEE Transactions On Parallel And Distributed Systems, vol.27, pp.2155-2167, July 2016.

[16] Meng Zhang, Jesse D.Bingham, John Erickson, and Daniel J. Sorin, " PVCoherence: Designing Flat Coherence Protocols for Scalable Verification," IEEE Micro, vol.35, pp.84-91, 2015.

[17] Abdullah Kayi, Olivier Serres, and Tarek El-Ghazawi, "Adaptive Cache Coherence Mechanisms with Producer–Consumer Sharing Optimization for Chip Multiprocessors," IEEE Transactions on Computers, vol.64, pp.316-328, February 2015.

[18] Hesham Altwaijry and Diyab S.Alzahrani, " Improved-MOESI Cache Coherence Protocol," Arabian Journal for Science and Engineering, Springer, vol.39, pp.2739-2748, April 2014.

[19] Christian Fensch, Nick Barrow-Williams, Robert D. Mullins, and Simon Moore, "Designing a Physical Locality Aware Coherence Protocol for Chip-Multiprocessors IEEE Transactions on Computers, vol.62, pp.914-928, 2013.

[20] Taeweon Suh, Hsien-Hsin S. Lee, and Douglas M. Blough, "Integrating Cache Coherence Protocols for heterogeneous Multiprocessor Systems, Part 1," IEEE Micro, vol.24, no.4, pp.33-41, 2004.

[21] Daniel J. Sorin, Manoj Plakal, Anne E. Condon, Mark D. Hill, Milo M.K. Martin, and David A. Wood, "Specifying and Verifying a Broadcast and a Multicast Snooping Cache Coherence Protocol," IEEE Transactions On Parallel And Distributed Systems, vol.13, pp.556-578, June 2002.