Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume12, Issue 5, July 2021: 3206-3216
Research Article

FPGA Implementation of Snoopy Bus based Cache Coherence Protocols for Dual Processor System

Sambu Navya[1], Dr. Y. Padma Sai[2], K. Swetha Reddy[3]

Abstract*:* When designing of Shared Memory Multiprocessor systems Cache Coherence becomes the great challenge to deal because incoherence may occur when the processors are working on the same data without any coordination. This coherence can be brought by using cache coherence protocols which are finite state machines which manages the cache and memory. This Paper aims at introducing cache coherence in details and providing a performance analysis of some of the cache coherence protocols. Thus this work is majorly focusing on implementation of MSI, MESI and MOESI cache coherence protocols using Xilinx ISE on FPGA and the way coherence protocols are designed are analysed perfectly.

***Keywords* -** Cache Coherence; MSI; MESI; MOESI; Verilog HDL; Artix-7 FPGA.

1. Introduction

Shared memory multiprocessor systems are emerging so rapidly to perform the computations at a faster rates. Processor speed is also increasing in order to meet the performance requirements of computations but the memory speed is not increasing at the same pace. In order to meet the demands of computations each processor in the multiprocessor system have its own private L1 Cache to reduce the memory access time of the processor.

In multiprocessor system cache coherence problem may occur when there is no proper coordination between the caches of the each Processor. data inconsistency may occur when there is no proper synchronization between caches. This leads to cache coherence problem and this is tackled by cache coherence protocols. To maintain the data consistency a set of rules are implemented in the system called as cache coherence protocols. In this paper three snoopy based cache coherence protocols are implemented are MSI, MESI, MOESI.

[1]VNR Vignana Jyothi Institute of Engineering and Technology Email: sambunavya@gmail.com
[2]VNR Vignana Jyothi Institute of Engineering and Technology
[3]VNR Vignana Jyothi Institute of Engineering and Technology

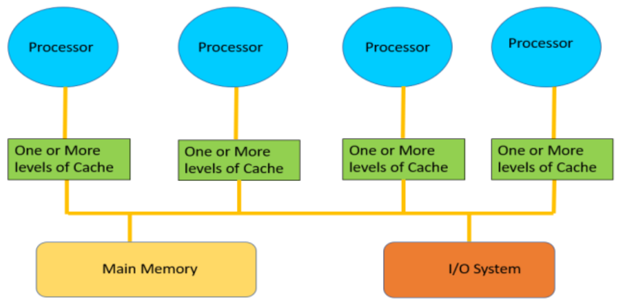

Processor
Processor
Processor
Processor
One or More levels of Cache
One or More levels of Cache
One or More levels of Cache
One or More levels of Cache
Main Memory
I/O System

Figure 1: Shared Memory Multiprocessor Architecture.

Figure 1 Shows the shared memory multiprocessor architecture where each processor consists its own independent cache and one main memory and I/O system.

2. Literature Survey

Zainab, Michael [2] describes only two snoopy based cache coherence Protocols. i.e., MSI, MESI. Kaushik Roy, Pavan Kumar S.R[3] have done comparative studies of cache coherence protocols but they have not implemented cache coherence protocols. Neethu, Geeta[4] have studied simulation based performance study of cache coherence protocols in Gem-5 Simulator. Sravanthi, Rajashekara[5] have implemented MESI Protocol. Daman, Sulochana[6] implemented cache coherence protocols in Multiprocessor systems. Liu, Hao [7] proposed a new type of L2 cache in order to combat the high power usage of the conventional L2 cache. Suamher[8] described the snoopy and directory based cache coherence protocols. Amit D.Joshi[9]acknowledged snooping Cache Coherence Protocols are suitable for multiprocessor architectures. Mays K. Faeq, Safa [10] have implemented only MSI Protocol in the multiprocessor system. Danko, Sinisa[11] have done Time domain performance evaluation of adaptive hybrid cache Coherence Protocol. Li, Sizhao [12] work explored the cache access patterns of a GPU by running various benchmarks on both NVIDIA and AMD architectures, and concluded GPU handles cache coherence much better than CPUs. Ibrahim A.Amory, Ahmed H.Ahmed [13] have implemented only MESI Protocol. Bijo, Shiji [18] Discussed parallel execution on multicore architectures with multilevel caches. This Literature survey motivated me to design of snoopy bus based cache coherence protocol for multiprocessor architectures.

3. Proposed Methodology

In multiprocessor systems, each processor has its own local cache. These caches can have different values for the same address in main memory depending on when the data was read from memory. For example, if two processors load the same value from memory and both modify it before storing it back to the memory, one of the values will be overwritten and the other value is lost. This problem is known as cache incoherence. There are multiple different protocols that attempt to keep coherency between the caches so when one processor modifies a value, the change is propagated through the rest of the system without wasting time for accessing the main memory.

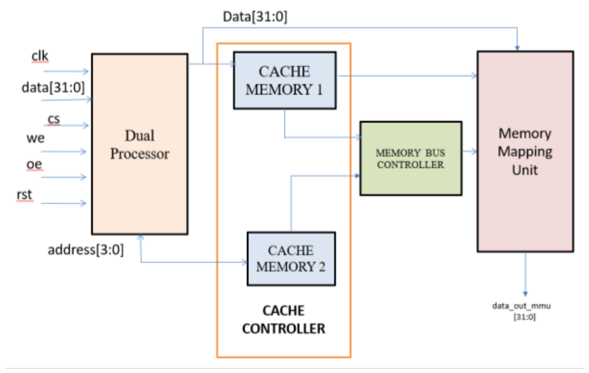


Figure 2: Proposed System for Cache Coherence in Multiprocessor System.

In figure 2 the chip select(cs), write enable(we),clk, rst, output enable (oe), data, address are the input signals applied to the processor. When the write enable is high the data and address is written to cache memory 1 and cache memory2 respectively. Cache memory1 is specifically designed for the purpose of storing the data. Sometimes, processor needs to perform operation immediately. so in that particular situations, if the processor looks for the data and task address in main memory, it will takes more time for performing the operation. To solve this problem, the data output from the processor will be applied as input to cache memory. It will stores the immediate version of data inputs, so the processor can perform the task very immediately. Thus, in order to perform the parallel operation, all the address inputs given by the user must be stored. For this purpose, the input addresses are going to store into the Cache memory2 for faster retrieval of data.

Generally, the results synchronization issues across the processor, to perform the Task based on the Cache-1 data output to the cache 2 address output. Thus, to maintain the synchronization between the two cache memories, the memory bus controller mechanism will be useful. The direct cache memory 1 output and memory bus controller output will be applied as the input to the memory mapping unit. Generally, the memory mapping unit is consisting of the real time processor, which consisting of multi-level memory to perform the various tasks. Thus, here the memory mapping unit is used to perform the selection of cache memory 1 output. If any interrupt generated due to the cache coherence protocol, then according to the interrupts, it will selects the data and results the outputs as the final data out.

A. Cache Coherence Protocols:

Cache coherence protocols are deployed in multiprocessor systems for maintaining coherency across all the caches for consistent view of shared data among the processors. Snoopy and directory based mechanisms are mainly used for maintaining cache coherency each having its own pros and cons.

Snooping protocols are faster since each cache status is monitored through bus but it is not scalable and it requires more bandwidth if the number of processors increase. Directory based protocols are scalable since it maintains separate directory for maintaining each cache status in the system but the disadvantage is longer latencies.

Snooping protocols were the first widely deployed class of coherence protocols and offer several attractive features such as low-latency transactions and a simple design. In such protocols, all the coherence controllers snoop the requests and process them in the same order. Thus, they all observe the same scenario and they can update correctly their finite state machines. Three widely used snooping protocols are MSI, MESI, MOESI.

a) MSI Protocol: MSI Protocol is the classic and basic cache coherence protocol for write back caches.MSI stands for Modified, Shared, Invalid. Each cache block is maintained one of the three states for maintaining coherency in the system.

Modified: The block is valid, exclusive, owned, may be dirty, and may be written or read. The cache has the only valid copy of the block and it is potentially stale at the memory. The cache is responsible for requests for the block.

Shared: The block is valid but not exclusive, not dirty, not owned and is read-only. The other caches may hold valid, read-only copies of the block.

Invalid: The block is invalid. Either the cache does not hold the block or it holds a stale copy that it may not read or write. Figure 3 represents the MSI Protocol State diagram.

Processor side requests for the cache includes:

PrRd: Processor request to read a cache block.

PrWr: Processor request to write a cache block.

Bus requests for the cache are:

BusRd: When a read miss occurs in a processor's cache, it sends a BusRd request on the bus and expects to receive the cache block in return.

BusRdX: When a write miss occurs in a processor's cache, it sends a BusRdX request on the bus which returns the cache block and invalidates the block in the caches of other processors.

BusUpgr: When there's a write hit in a processor's cache, it sends a BusUpgr request on the bus to invalidate the block in the caches of other processors.

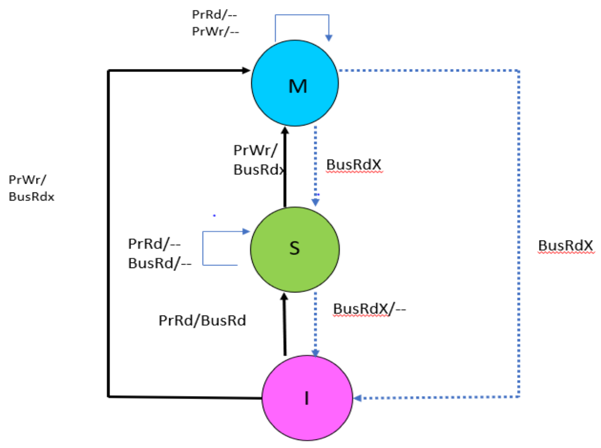


Figure 3: MSI State Transition Diagram

b) MESI Protocol*:* MESI protocol is wite invalidated cache coherence protocol mainly adopted for write back caches. MESI protocol is also known as Illinois protocol named after the Illinois university where it is developed.

In MESI Protocol an Extra State is added to MSI Protocol called as Exclusive state which reduces the Transitions from "invalid" to "modified" state from MSI Protocol.

Exclusive: A block is exclusive when it is the only privately cached copy of that block in the system. The importance of the exclusive state is that local Cache can modify its data alone without need for Snooping from other Caches.

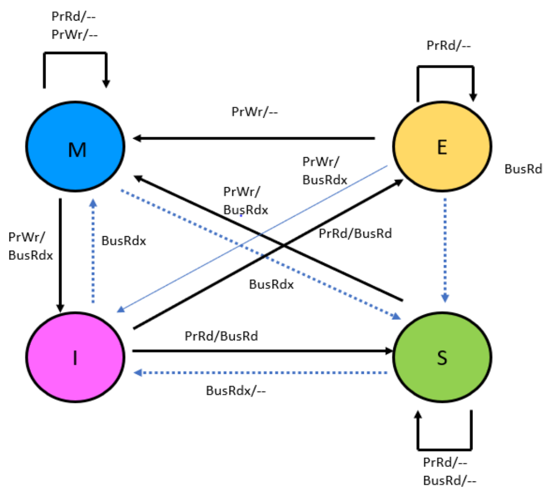


Figure 4: MESI State Transition Diagram

The advantagaes of MESI Protocol over MSI protocol is the extra state added called as exclusive state which reduces the bus transistions when the read miss occurs in the cache.MESI protocol is faster when compared to MSI particularly when working on the serial application by the processor. No cost change will be there since MSI and MESI states are encoded with 2 bits.

c) MOESI Protocol: In MOESI an extra state called as Owned State to MESI Protocol is added to improve the MESI which reduces the memory access. Figure 5 represents the MOESI State Transition Diagram of MOESI.

Owned state: In the multiprocessor system, A block is owned by a cache controller if it is responsible for responding to coherence requests for that block. This block cannot be evicted without giving the ownership to another coherence controller.

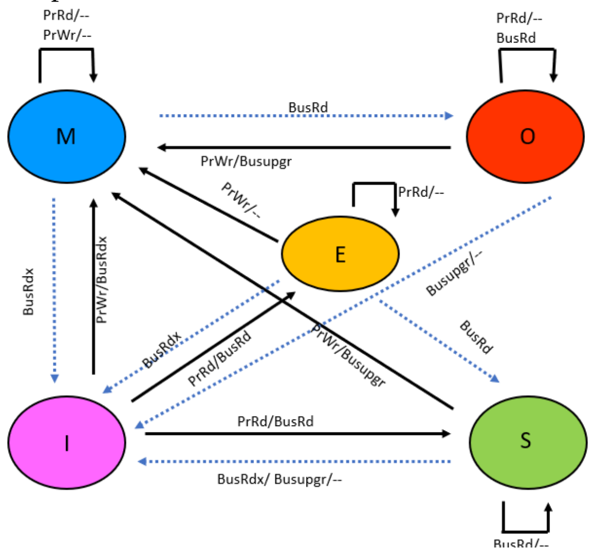


Figure 5: MOESI State Transition Diagram.

The Owned state in MOESI protocol overcomes the writing of dirty cache line back to main memory. The owned state allows directly to access the modified data to the processor. When the processor wants to write the data to its cache line with owned state it has to send bus request to invalidate the cache line which is having same block or asks to update their cache contents with new data.

4. Results

In this Paper MSI, MESI, MOESI Cache Coherence Protocols of Shared Memory dual processor system is designed in Xilinx and implemented on FPGA.

i) Design of Protocols:

State machines of snoopy bus based cache coherence protocols are designed in Verilog HDL and simulated in Xilinx. The various input signals are Processor read(PrRd), Processor Write(PrWr), Bus Read(BusRd), Bus upgrade(BusUpgr) determines the Cache states and also Output signals Memory Read(mem_rd), Memory write(mem_wr), Memory chip select(mem_cs) determines the possible operations that can be performed on the cache states.

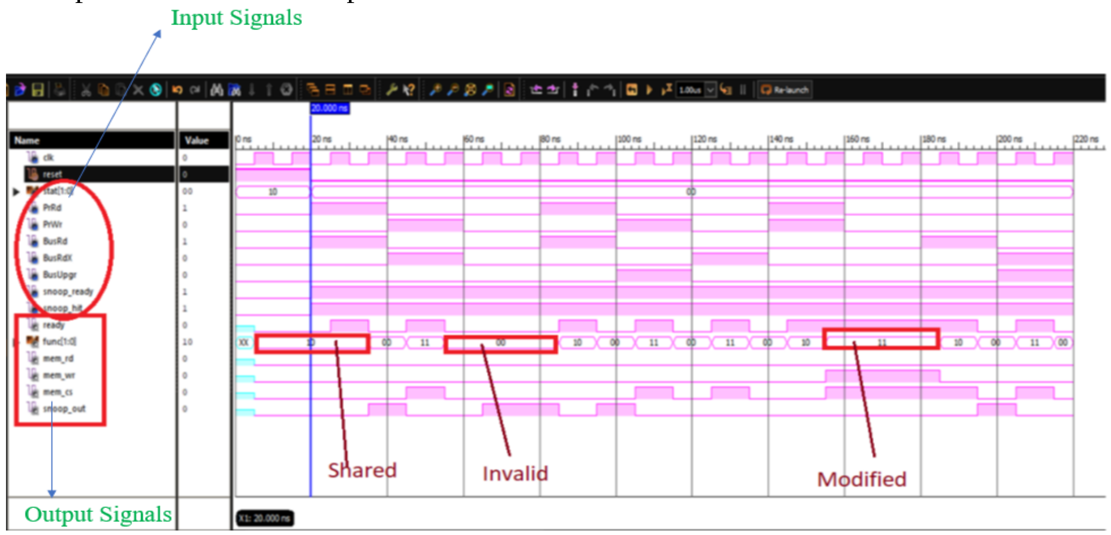


Figure 6: Simulation Waveform of State Transition of MSI Protocol

Figure6 Represents the Simulation Waveform of MSI Protocol. It shows States of cache block States are changed according to input signals.

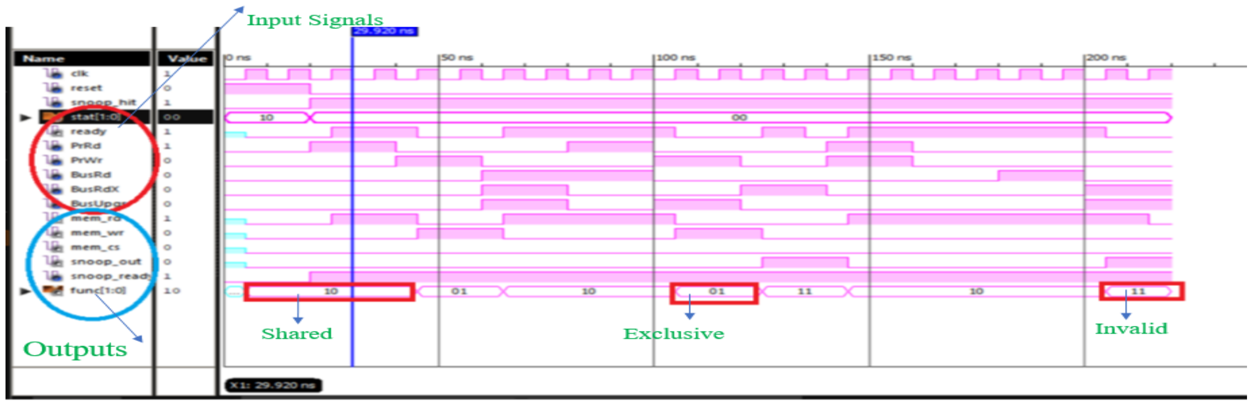


Figure 7: Simulation Waveform of State Transition of MESI Protocol

Figure 7 Represents The Simulation Waveform of MESI Protocol.

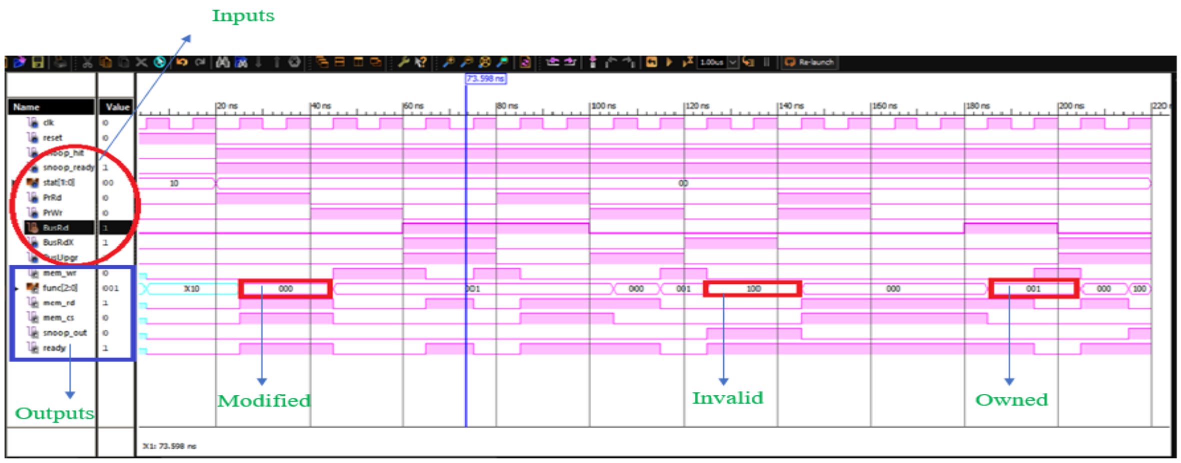


Figure 8: Simulation Waveform of State Transition of MOESI Protocol

Figure 8 Represents the Simulation Waveform of MOESI Protocol.

ii) Implementation of Protocols:

MSI, MESI, MOESI Snoopy bus based Cache Coherence Protocols are Implemented in the Proposed System as shown in Figure1. All the Individual modules of Proposed System are designed separately in Xilinx.

Testbenches are designed to verify the functionality of Each Protocol in the Proposed System of Cache Coherence Multiprocessor System. Each Module is instantiated in the top level module by calling each Module in the top level module according to their functionality. Here in the Proposed System clk, address, data, cs(chip select) ,we(write enable) ,oe(output enable) are the input signals, remaining all are output signals. Data out processor should be monitored with respect to the address input. Initially, when we=1 then all the data input will be stored into RAM. Then memory based error checking operation performs using rollbacks. Finally, ram_oe will be activated. Then Data out of processor signal will generates. Data out of cache should be monitored with respect to the address out of cache. After error correction,cam1_oe becomes 1, then data out will be generated. The valid_f flag becomes active high, whenever error occurred, it will be auto corrected by using the proposed system.

Figure 9,10,11 are the simulation waveforms of Implementation of MSI, MESI, MOESI Protocols in the Proposed system.

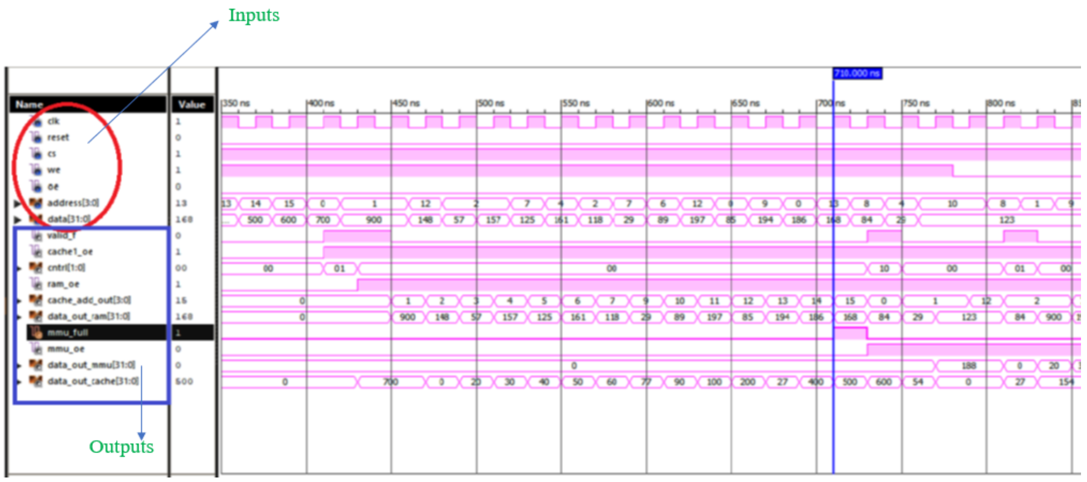


Figure 9: Implementation of MSI Protocol

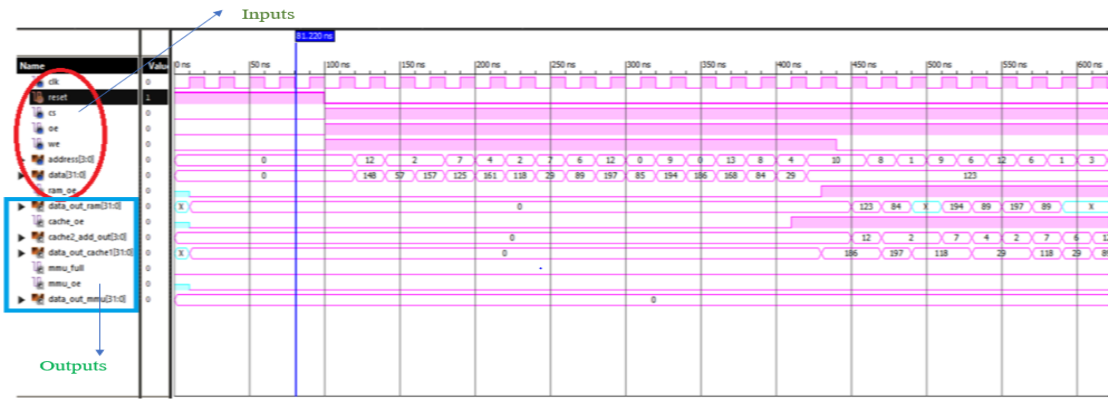


Figure 10: Implementation of MESI Protocol

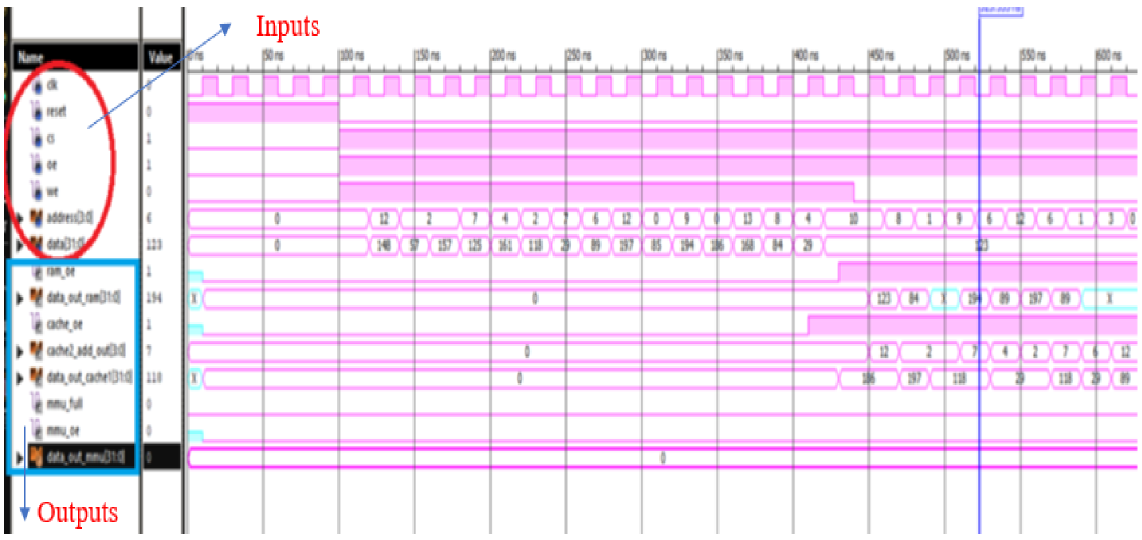


Figure 11: Implementation of MOESI Protocol

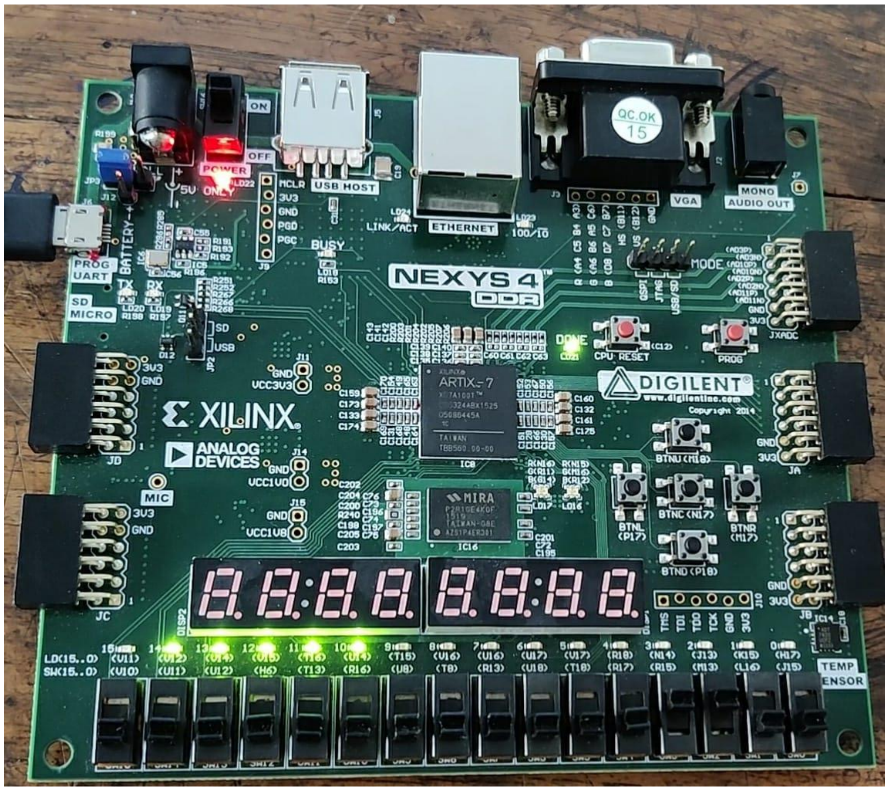
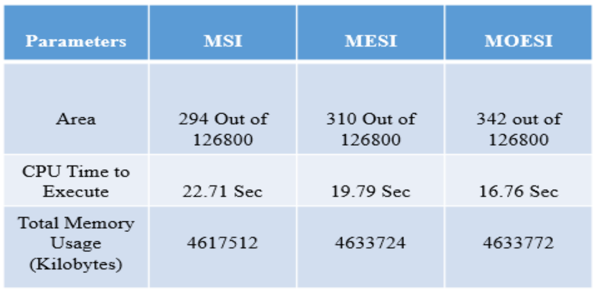


Figure 12: Implementation of Cache Coherence Protocol in ARTIX-7 FPGA

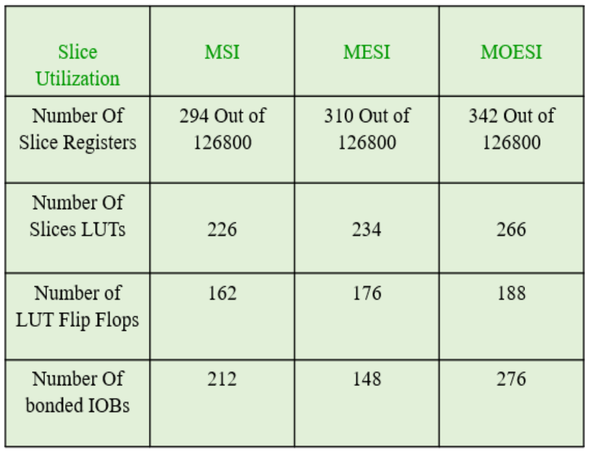
Figure 12 Represents the FPGA Implementation of Cache Coherence Protocol. Above figure Represents The one of the state in MESI Protocol. Likewise all The states in Cache Coherence Protocol are implemented in FPGA Board.

TABLE 1: COMPARISON OF CACHE COHERENCE PROTOCOLS IN TERMS OF ITS PERFORMANCE



Parameters	MSI	MESI	MOESI
Area	294 Out of 126800	310 Out of 126800	342 out of 126800
CPU Time to Execute	22.71 Sec	19.79 Sec	16.76 Sec
Total Memory Usage (Kilobytes)	4617512	4633724	4633772

TABLE 2: COMPARISION OF MSI, MESI, MOESI PROTOCOLS



Slice Utilization	MSI	MESI	MOESI
Number Of Slice Registers	294 Out of 126800	310 Out of 126800	342 Out of 126800
Number Of Slices LUTs	226	234	266
Number of LUT Flip Flops	162	176	188
Number Of bonded IOBs	212	148	276

4. Conclusion

In this paper MSI, MESI, MOESI Cache coherence protocols of Dual Processor system is designed, verified in Xilinx and implemented in ARTIX7 FPGA. Their Area, delay performance analysis is carried out. These results shows us that MOESI Protocol actually bring significant performance improvements in the Shared Memory Multiprocessor System.

References

[1]. Vijay Nagarajan, Daniel J.Sorin, Mark D. Hill, David A.Wood. " A primer on Memory Consistency and Cache Coherence. "Morgan and Claypool Publishers.
[2]. Zainab Al-Waisi and Michael Opoku Agyeman. "An Overview of On-Chip Cache Coherence Protocols." 2017 Intelligent System Conference.
[3]. Kaushik Roy, Pavan Kumar S.R, Meenatchi S. "Comparative Study On Cache Coherence Protocols." 2016 IOSR Journal of Computer Engineering.
[4]. Neethu Bal Mallya, Geeta Patil and Biju Raveendran. " Simulation Based Performance Study of Cache Coherence Protocols." 2015 IEEE International Symposium on Nanoelectronics and Information Systems.
[5]. Attada Sravanthi, Ch.Rajasekhara Rao, K. Krishnam Raju, L. Rambabu. "Implementation of MESI Protocol in Verilog." 2019 International Research Journal of Engineering and Technology.
[6]. Daman Preet Kaur, V. Sulochana. "Design and Implementation of Cache Coherence Protocol for High-Speed Multiprocessor System." 2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems.
[7]. Liu, Hao, Quentin L. Meunier, and Alain Greiner. "Decoupling Translation Lookaside Buffer Coherence from Cache Coherence." *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017.
[8]. Samaher AI- Hothali, Safeeullah Soomro, KhuraamTanvir, Ruchi Tuli. " Snoopy and Directory Based Cache Coherence Protocols: A Critical Analysis." Journal of Information & Communication Technology. Vol.4, No.1, (Spring 2010) 01-10.
[9]. Amit D.Joshi, Satyanarayana Vollala, B. Shameeda Begum, N. Ramasubramanian. " Performance Analysis of Cache Coherence Protocols for Multi-Core Architectures: A

System Attribute Perspective.” AICTC 16: Proceedings of the International Conference on Advances in Information Communication Technology and Computing 2016.

[10]. Mays K. Faeq, Safaa S.omran. “ MSI Protocol for Multicore Processors Based on FPGA.” International Journal of Engineering Science Invention (IJESI) 2020.

[11]. Danko Ivosevic, Sinisa Srbljic, Vlado sruk. “Time Domain Performance Evaluation of Adaptive Hybrid Cache Coherence Protocols.” IEEE 10th Mediterranean Electroteechnical Conference.

[12]. Li, Sizhao, and Donghui Guo. "Cache coherence scheme for HCS-based CMP and its system reliability analysis." IEEE Access 5 (2017): 7205-7215.

[13]. Amory, Ibrahim A., Ahmed H. Ahmed, and Zahraa Hasan. "MESI protocol for multicore processors based on FPGA." Periodicals of Engineering and Natural Sciences (PEN) 9.1 (2021): 80-89.

[14]. Chakraborty, Bidesh, Mamata Dalui, and Biplab K. Sikdar. "Design of a reliable cache system for heterogeneous CMPs." Journal of Circuits, Systems and Computers 27.14 (2018): 1850219.

[15]. Kaushik, M. Hassan and H. Patel “Designing Predictable Cache Coherence Protocols for Multi-Core Real-Time Systems.” IEEE Transactions on Computers, 2020.

[16]. R. Rodrigues, I. Koren, and S. Kundu, “A mechanism to verify cache coherence transactions in multicore systems,” in 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Oct 2012.

[17]. Kabadi, Dr, and G. Mohan. "A Comprehensive Study on Design Consideration of Multi Core Processors." (2020).

[18]. Bijo, Shiji, et al. "A formal model of parallel execution on multicore architectures with multilevel caches." International Conference on Formal Aspects of Component Software. Springer, Cham, 2017.

[19]. Agarwal, Sukarn, and Hemangees K. Kapoor. "LiNoVo: Longevity Enhancement of Non-Volatile Last Level Caches in Chip Multiprocessors." 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2020.

[20]. sLyu, Yangdi, et al. "Directed test generation for validation of cache coherence protocols." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.1 (2018): 163-176.