Periodicals of Engineering and Natural Sciences ISSN 2303-4521
Vol. 9, No. 1, January 2021, pp.80-89

MESI protocol for multicore processors based on FPGA

Ibrahim A. Amory[1], Ahmed H. Ahmed[2], Laith F. Jumma[3]
[1] Al-Esraa University College
[2] Al-Farahidi University
[3] Al-Esraa University College

ABSTRACT

In modern techniques of building processors, manufactures using more than one processor in the integrated circuit (chip) and each processor called a core. The new chips of processors called a multi-core processor. This new design makes the processors to work simultaneously for more than one task or all the cores working in parallel for the same task. All cores are similar in their design, and each core has its own cache memory, while all cores share the same main memory. So, if one core requests a block of data from main memory to its cache, there should be a protocol to declare the situation of this block in the main memory and other cores. This is called the cache coherency or cache consistency of multi-core. In this paper a special circuit is designed using VHDL coding and implemented using ISE Xilinx software, one protocol was used in this design, the MESI (Modify, Exclusive, Shared and Invalid) protocol. Test results were taken by using test bench, and showed all the states of the protocols are working correctly.

Keywords: Multicore Processor, MESI protocol, VHDL, MIPS, FPGA.

Corresponding Author:

Ibrahim A. Amory
Department of Medical Instrumentation Engineering
Al-Esraa University College, Iraq
E-mail: ibrahim.a.amory@esraa.edu.iq

1. Introduction

In modern techniques of computers, the multi-core processor is used. All processors or cores will share the same main memory of the system for simple programming model. Hence all cores will share through single address space the shared memory. All cores will work for the same task, so if one core writes on some blocks from the shared memory it must be some rules must be **existing** to tell the other core not to use that blocks of data until an update must be done. The cache coherence protocol in the term used to confirm that all the used information **is** write [1].

Shared memory multi-core needs to be coherent and consistent with data [2]. In recent years multi-Core are gaining more importance as they have better performance and reliability than single-core system figure 1. Multi-core with shared memory is being used in the today's computers [3]. The current hardware world is dominated by the multi-cores or many-cores [4]. As the trend shifts from single-core to multi-core processors for tuning up the performance.

Nowadays, chip multi-core is the main trend in designing the CPU for high performance devices. This originates from the fact that the single core chip reaches the limitation of execution speed because of the heat and power dissipation issues [5-8]. Moreover, modern technologies support millions of transistors to be integrated in one chip which eases the design of multi-core on chip in terms of area. In fact, several multi-cores have been commercialized in the market [9-10].

The caches in shared memory multiprocessors are used to improve performance by reducing the processor's memory access time. Unfortunately, caching introduces the cache coherence problem. Early shared-memory machines left it to the programmer to deal with the cache coherence problem, and therefore these machines were considered difficult to program [11].

To solve the problem of cache coherence, designers used hardware circuit that implemented in the processor. This hardware will snoop all the activities of all the cores and uses a certain mechanism which called cache coherence protocol to put an appropriate sign for each action by waiting the bus [12].

In this thesis, a pre-designed MIPS (Microprocessor without Interlocked Pipeline Stages) type single core processor is used. This processor is used to build another processor similar to it so that to have two cores and a

cache was built for each processor. Then cache was chosen with capacity (32-bits) and direct mapped type for ease of design. Then building a circuit of coherence controller in order to control the reading and writing processes for processors when reading and writing from the shared memory of each of the processors. A link to all the designed parts, and several programs were written for the purpose of operating and examining the three types of coherency protocol MESI [13][14].

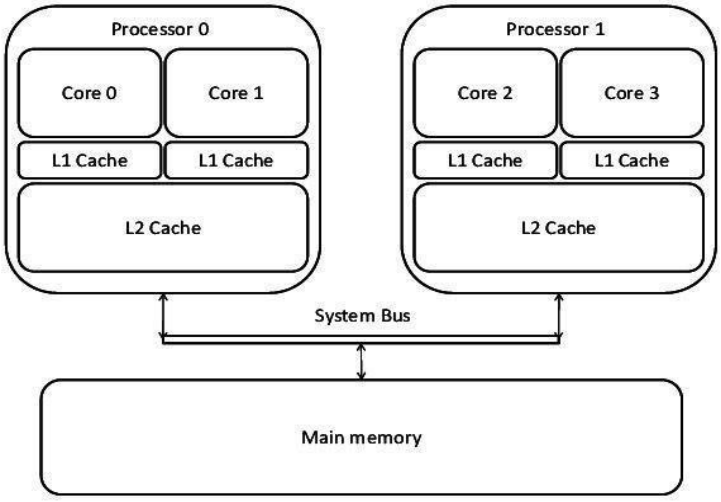

Figure 1. Dual multicore processor.

2. Research method

2.1. Differences of cache coherency protocols

This section summarizes the differences between MSI, MESI, and MOESI cache coherency protocols. MSI is the basis of three other protocols. When using MSI, the cache line is in one of three modes: modified, shared, or invalid. The MESI protocol will add additional exclusive status the benefit of adding the Exclusive state is to reduce the number of broadcasts when writing to a line that is present in only one of the caches. Because an Exclusive rows reside in a single cache, so there is no need to broadcast an invalidation signal on write [15][16]. Conversely, if you write to a shared line, you need to broadcast the invalidation signal because the remote cache may contain a shared copy [17][18].

The MOESI protocol has the advantage of adding both exclusive and owner states, reducing both the number of broadcasts and the number of rewrites. Table 1 show the different between three protocols [19][20].

Table 1. The difference between cache coherence protocols

MSI protocol	MESI protocol	MOESI protocol
MSI is basis of three state(Modified(M),Shared (S), and Invalid (I)).	MESI is basis of four states(Modified(M),Exclusive (E),Shared (S), and Invalid (I)).	MOESI is basis of five states(Modified(M),Owned (O),Exclusive(E),Shared (S), and Invalid (I)).
multiple copies of the block at the same time can do and transition from Shared to Modify can be done without reading data from the cache.	Exclusive (E) added to reduce the number of bus messages sent out for invalid to modified transition.	Owned (O) added to avoid the need of copying back to main memory (write update).
Area utilization of MSI protocol is few by number of use register and flip-flob.	Area utilization of MESI protocol is more as compared to MSI protocol.	Area utilization of MOESI protocol is more as compared to MESI and MSI protocol.

2.2 FPGA (Field Programmable Gate Array)

The FPGA is an integrated circuit consisting of a grid matrix that can be programmed by the user "in the field" without the use of expensive equipment [15]. An FPGA consists of a set of programmable logic gates and rich interconnect resources from which complex digital circuits can be implemented as shown in figure 2.
It consists of three main parts:
1. Configurable Logic Blocks — which implement logic functions.
2. Programmable Interconnects — which implement routing.
3. Programmable I/O Blocks — which connect with external components.

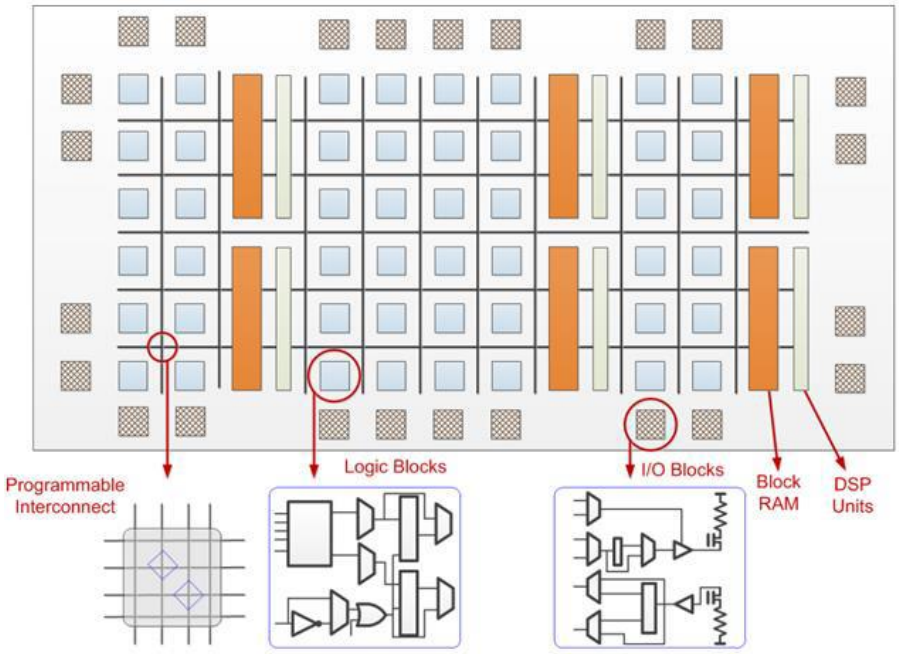

Figure 2. Internal component of FPGA

2.3 Design and implementation of cache coherence protocols

Design and implementation dual-core MIPS type processor in which each processor contains its cache memory and the two processors used the same main memory. Figure 3 shows a block diagram for the estimated design of dual-core MIPS processor [21].

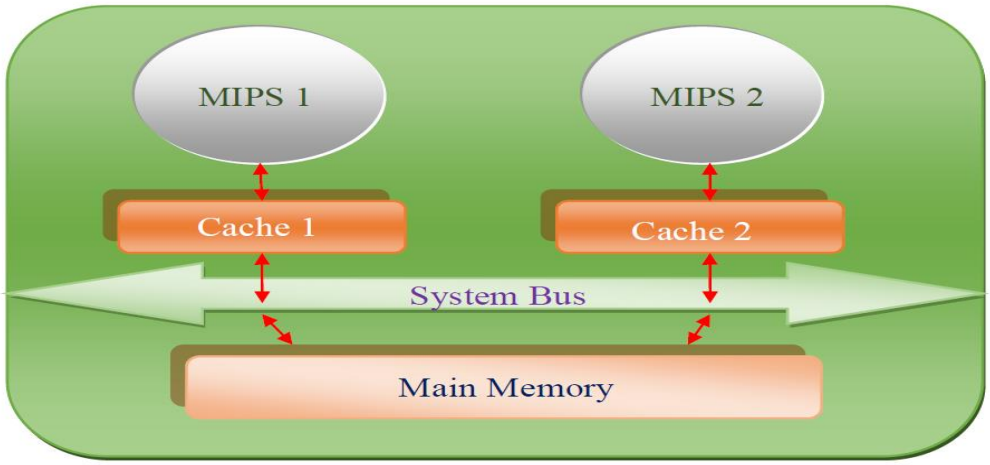

Figure 3. Architecture of dual microprocessor

Design and implementation a cache controller circuit and link it to the main processors and memory in order to perform the process of controlling and monitoring the required addresses [22].

2.4 Requirements for proper coherence protocol

For any good coherence protocol there should be some important features need to be followed.

1. Latency: the latency should be low for cache-to-cache transfer.
2. Bandwidth: it must be avoided for bus-communication to provide efficient bandwidth.

Snoopy-based method transmits messages on buses, thus supporting low-delay, cache-to-cache transfers. In contrast, directory-based protocols send requests to destinations and wait for confessions that require an extra clock cycle. Second, it refuses to rely on bus-like architecture to limit the integration of more cores into the system. The directory-based protocol has the advantage of stabilizing a large number of cores that communicate point-to-point. Third, bandwidth performance can only be reduced to some extent. Should reduce the bandwidth to avoid interconnected conflicts as this affects the performance of the system. Figure 4 shows the need for coordination. The width of the triangle represents several desired functions. Each feature is associated with the same protocol.

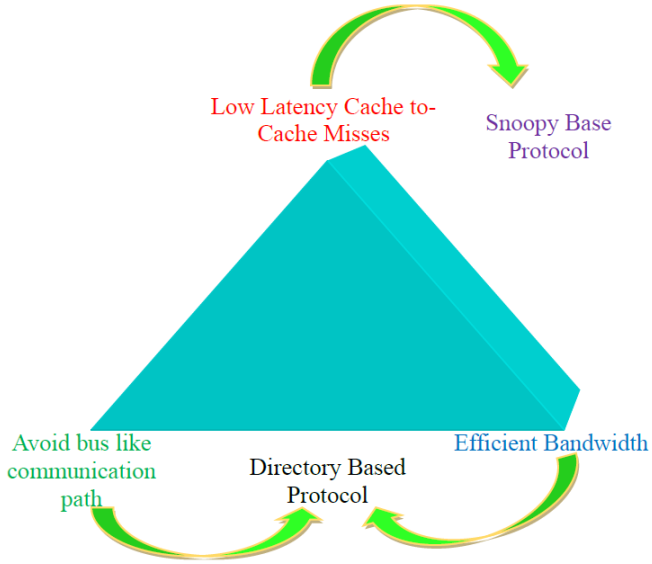

Figure 4. Several desire function

2.5 Snoopy based coherence protocol

Snoopy and directory methods are standard for maintaining catch coherence in multi-core systems. Both methods have their merits and demerits. In today's researchers, bandwidth performance, low-delay cache-to-cache storage, and an integrated storage method that facilitates interconnection on a bus.

Snoopy is not suitable for large, scalable multi-core systems. In addition, the directory method incurs directory storage overhead and increases the latency of transfers between caches. Therefore, the evolution of cache coherency protocols has been used to achieve high performance, regardless of design limitations. First Protocol MSI (Modified (M), Shared (S) and Invalid (I)) This protocol was developed to MESI exclusive (E).

3. MESI Protocol

The MESI protocol has one more state than the previous protocols and this state called Exclusive (E) state. The action of this state and other states are given in Table 2. This new state is added in order to reduce the number of bus message .Because the Exclusive state means that the block is valid only in this cache and main memory and not valid in any other caches. This given a flexibility to the processor to modify its cache without a need to snoop other caches. Figure 5 shows the state transition diagram for the MESI protocol. The left side of the figure represent the processor requests and actions of the cache controller circuit, while the right part of the figure represents the bus requests and corresponding action. Table 2 shows the MESI state transfer, and Figure 6 shows a simple MESI state diagram with state transfer.

Table 2. The action of MESI State

States	Processors Load	Processors Store	Processors Eviction	Incoming Read Req.	Incoming Write Req
Modified	Hit	Hit	Write-back and Change to Invalid	Send data Write-back and Change to Shared	Send data and Change to Invalid
Exclusive	Hit	Change to Modified	Silent Evict and Change to Invalid	Send data Write-back and Change to Shared	Send data Hit and Change to Invalid
Shared		Change to Modified	Silent Evict and Change to Invalid	None	Change to Invalid
Invalid	Hit	Change to Modified	None	None	None

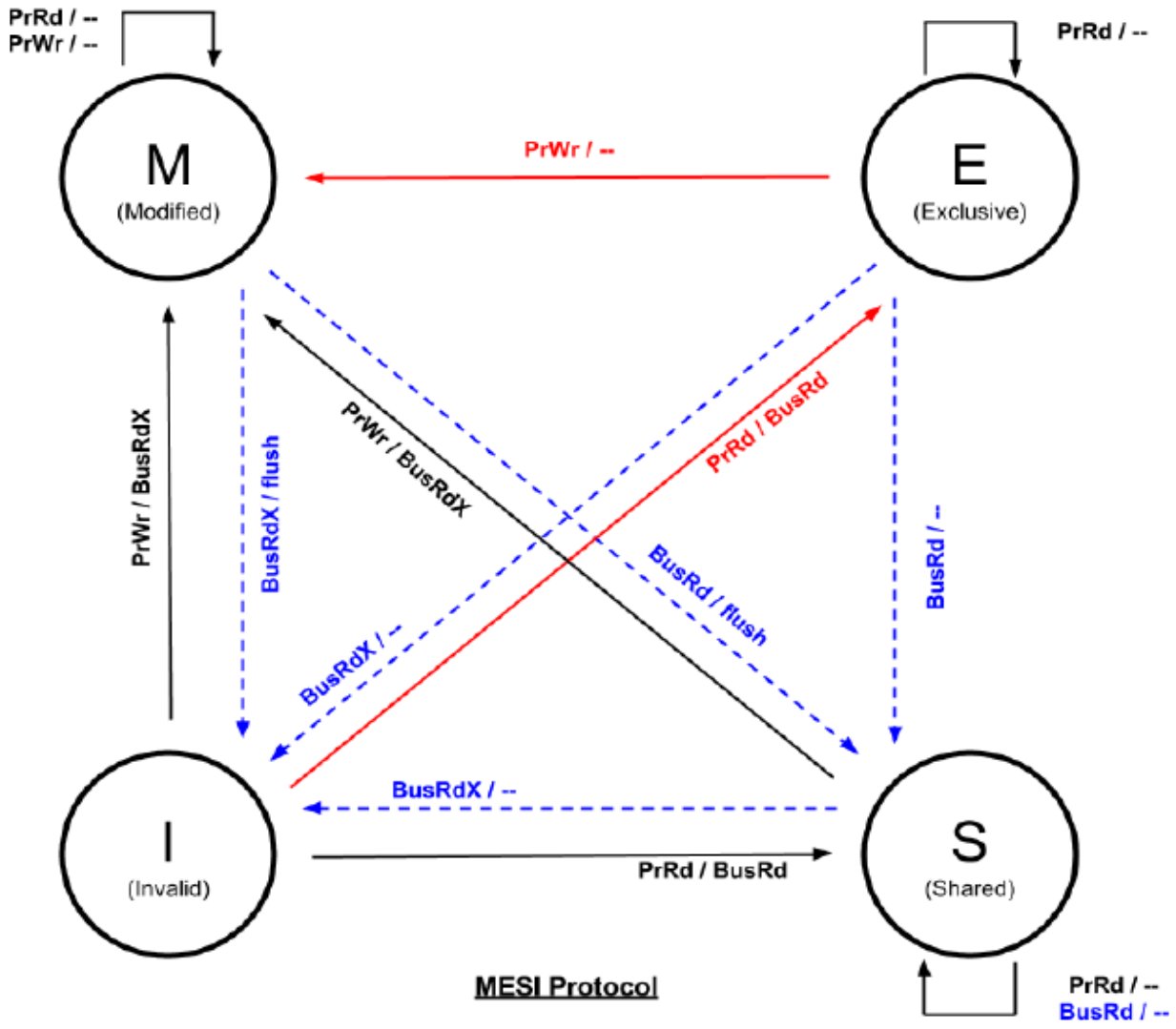

Figure 5. State transition diagram for the MESI protocol

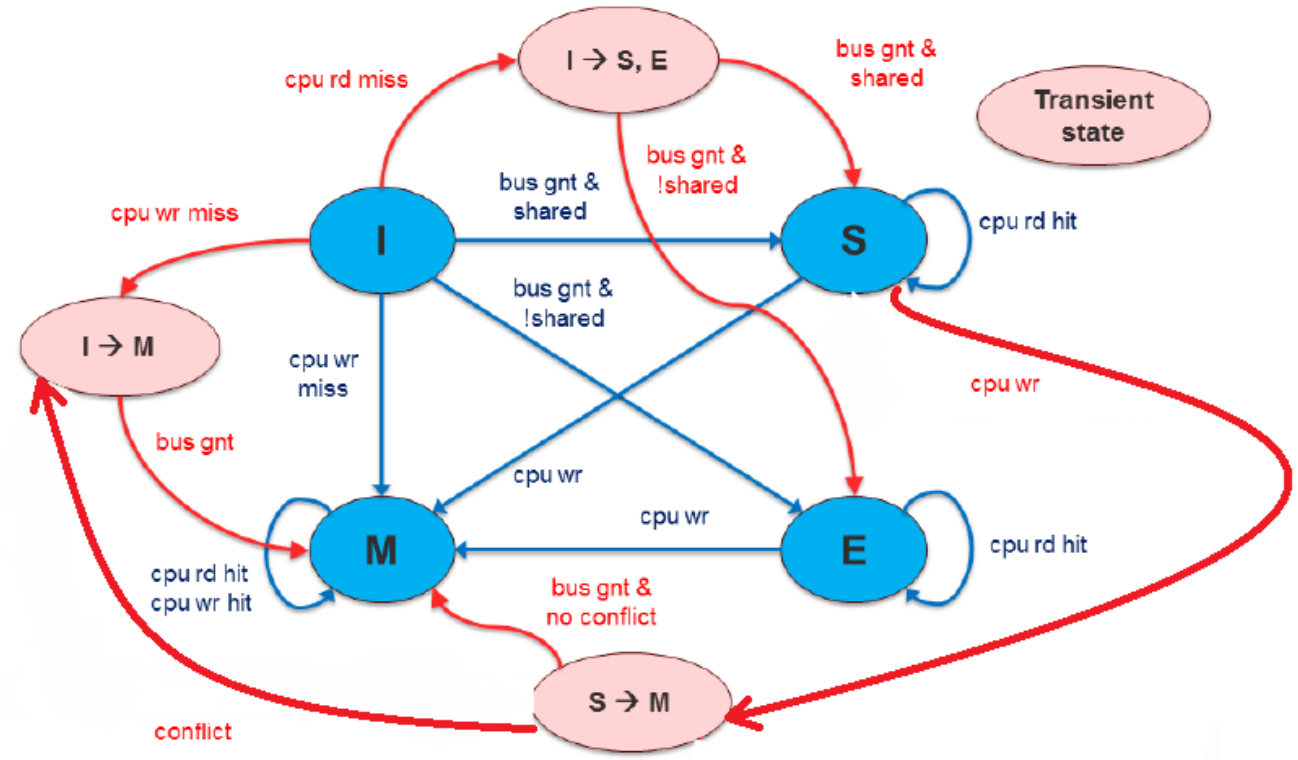

Figure 6. Simple MESI state diagram with state transfer

❖ Advantage of MESI protocol:

1. Differentiation between modified and shared state.

2. There can be multiple copies of the block at the same time.

3. The transition from Shared to Modify can be done without reading data from the cache.

4. Exclusive (E) added to reduce the number of bus messages sent out for invalid to modified transition.

❖ Disadvantage of MESI protocol:

1. Additional hardware is required to decide on a block transfer when a read request is received.
2. Whenever the "M" state changes to "S", the data needs to be written back to memory.
3. Offsets, when transitioning from "M" to "S" or "I". If the data does not need to be reused, the data converted into shared will be reused and rendered inappropriate.

4. Cache coherency controller design

In this work two multi-core MIPS1 and MPIS2 were designed and each with separate cache and both sharing the same main memory. A cache coherency controller is designed which consist of two parts, the coherency tag and coherence controller by using FSM (Finite State Machine). All these components are connected together on chips and have the main memory off chip. Figure 7 shows a block diagram for the cache coherency controller when connected to on chip with the two processors. The design has two caches, cache A and cache B. The caches is a directly mapped set association. Each cache contains eight sets, each containing four 32-bit data, a 26-bit tag, and a valid bit, update, dirty, and valid bit. Two processor components, MIPS1 and MIPS2, are used to perform write and read functions in the cache. Shared memory used in the design contains 32 entries. Bus controller is used to synchronize between different modules accessing the shared bus at the same time.

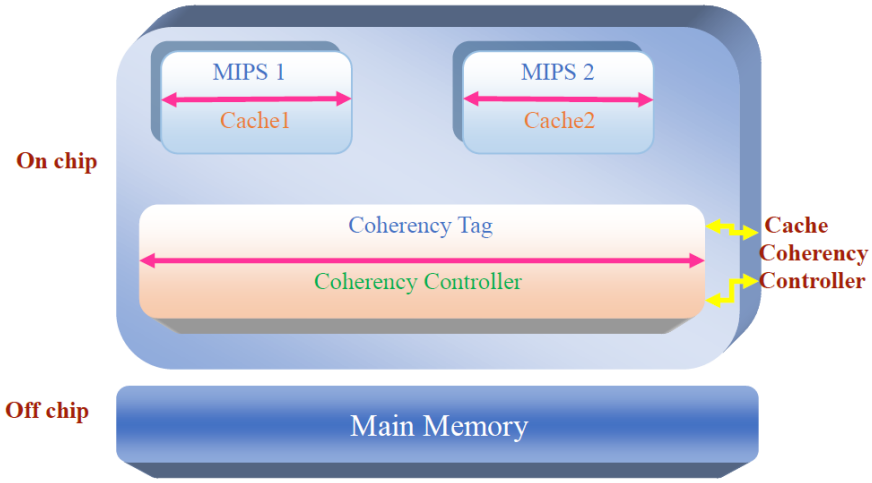

Figure 7. Block diagram for the cache coherency controller

Cache controller which decides this address exists in tag cache or not. If it exists then no memory access is needed, cache controller provides this data to processor from cache memory; if it does not exist then the cache controller fetches the data from main memory.

5. Cache coherence controller for MESI protocol

For the second type of protocol MESI, a 7- bit were used to indicate different states for MESI protocol, two bits for M (Modify), two bits for E (Exclusive), one bit for S (Shared) and two bits for I (Invalid). Figure 8 shows the coherency tag with 7-bits for MESI states. Table 3 shows MESI states. But a new state has been added called Exclusive state and the rest of states are as in MSI controller.
When Exclusive (01); this means that Cache line is the same as in main memory and in the cache of first microprocessor (MIPS1).
Table 3-4 shows the different sates of MESI Protocol. Only two states will be explained in this section:
1. For (St3) if (M=10, E=00, S=0, I=00) the OutMESI1 will be Exclusive in MIPS1 Hit read (Direct read).
2. For (St9) if (M=01, E=00, S=0, I=00) the OutMESI1 will be Not Exclusive in MIPS1 Hit write (Direct write).

Table 3. MESI states

M (*Modify*)	E (*Exclusive*)	S (*Shared*)	I (*Invalid*)
MODIFY			
00	NOT MODIFIED		
01	MODIFIED FIELD MP1		
10	MODIFIED FIELD MP		
EXCLUSIVE			
00	NOT EXCLUSIVE		
01	EXCLUSIVE FIELD MP1		
10	EXCLUSIVE FIELD MP2		
SHARED			
0	NOT SHARED		
1	SHARED		
VALID			
00	VALID BOTH		
01	NOT VALID MP1 (INVALID 1)		
10	NOT VALID MP2 (INVALID 2)		

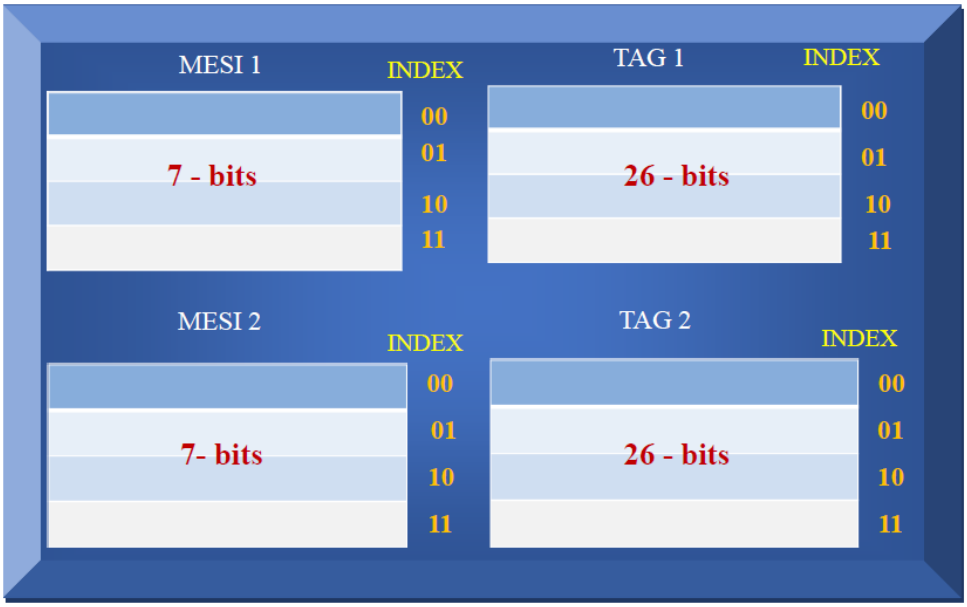

Figure 8. Coherency tag with 7-bits for MESI states

6. Simulation results of MESI protocol

For the second type of protocol MESI, a 7- bit were used to indicate different states for MESI protocol, two bits for M (Modify), two bits for E (Exclusive), one bit for S (Shared) and two bits for I (Invalid). But a new state has been added called Exclusive state and the rest of states are as in MSI controller.
When Exclusive (01); this means that Cache line is the same as in main memory and in the cache of first microprocessor (MIPS1).
Only two states will be explained in this section:
1. For (St3) if (M=10, E=00, S=0, I=00) the Out MESI1 will be Exclusive in MIPS1 Hit read (Direct read). Figure 9 shows the test bench of MESI protocol-MP1 Hit Read (Direct Read).

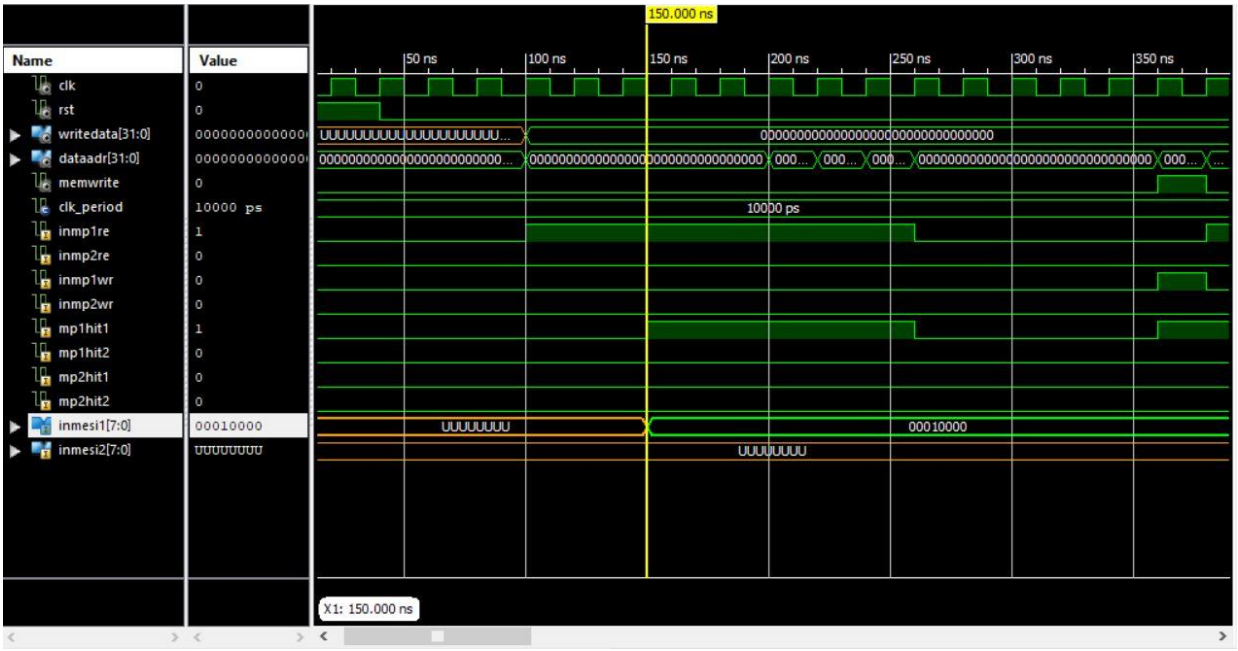

Figure 9. Test bench of MESI protocol-MP1 Hit Read (Direct Read)

2. For (St9) if (M=01, E=00, S=0, I=00) the OutMESI1 will be Not Exclusive in MIPS1 Hit write (Direct write). Figure 10 shows the test bench of MESI protocol-MP1 Hit Write (Direct Write).

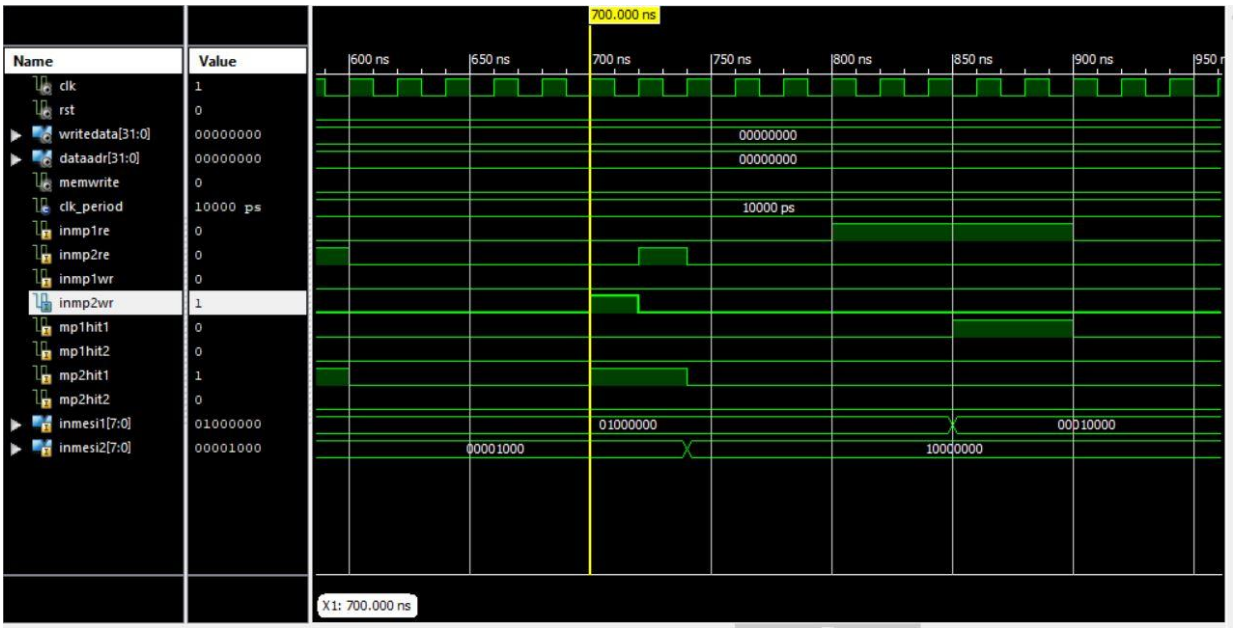

Figure 10. Test bench of MESI protocol-MP1 Hit Write (Direct Write).

7. FPGA device utilization

After all designs are synthesized successfully, Xilinx ISE Design Suite 14.1 software provides estimated values of the hardware amount that is needed to build each design. Table 4 shows the hardware amount of MESI design using Xilinx virtex 7.

Table 4. The hardware amount of MESI design using Xilinx virtex 7

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1264	54576	2%
Number of Slice LUTs	16319	27288	59%
Number of fully used LUT-FF pairs	1087	16496	6%
Number of bonded IOBs	68	296	22%
Number of BUFG/BUFGCTRLs	8	16	50%
Number of DSP48A1s	16	58	27%

8. Conclusion

The Cache Coherency Controller was built for MESI protocols. In this paper a special circuit is designed using VHDL coding and implemented using ISE Xilinx software. Test results were taken by using test bench, and showed all the states of the protocol are working correctly.

Several suggestions could be stated, these suggestions could be considered as the basis for further work. It is possible to improve compatibility protocols because each existing protocol has some limitations. Lower bandwidth usage, less network traffic, require major training. Off-chip communication must be reduced to reduce long latency for off-chip memory access.

References

[1] R. Fuchsen, "How to Address Certification for Multicore Based Ima Platforms: Current Status and Potential Solutions," Digital Avionics Systems Conference (DASC), 2010.

[2] P. S. a. W. Zhang, "WCET Estimation of Multi-Core Processors with the MSI Cache," *Proc. Work-in-Progress Session LCTES*, pp.17-20, 2013.

[3] D. J. Sorin, M. D. Hill, and D. A. Wood, A Primer on Memory Consistency and Cache Coherence, 1st ed. Morgan & Claypool Publishers, 2011.

[4] F. Pong and M. Dubois, "A new approach for the verification of cache coherence protocols," IEEE Trans. Parallel Distrib. Syst., vol. 6, no. 8, pp. 773–787, Aug. 1995.

[5] Wagner and V. Bertacco, "Caspar: Hardware patching for multicore processors," in 2009 Design, Automation Test in Europe Conference Exhibition, April 2009.

[6] D. Borodin and B. H. H. Juurlink, "A low-cost cache coherence verification method for snooping systems," in 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, Sep. 2008.

[7] F. Pong and M. Dubois, "The verification of cache coherence protocols," in Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures, ser. SPAA '93. New York, NY, USA: ACM, 1993.

[8] J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Dynamic verification of cache coherence protocols," in High Performance Memory Systems. Springer, 2004.

[9] DeOrio, A. Bauserman, and V. Bertacco, "Post-silicon verification for cache coherence," in 2008 IEEE International Conference on Computer Design, Oct 2008.

[10] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," in Proceedings of the 11th Annual Symposium on Computer Architecture, Ann Arbor, USA, June 1984, 1984.

[11] M. Kinsy, M. Pellauer, and S. Devadas, "Heracles: A tool for fast rtlbased design space exploration of multicore processors," in Proceedingsof the ACM/SIGDA international symposium on Field Programmable Gate Arrays, ser. FPGA '13. New York, NY, USA: ACM, 2013.

[12] 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.

[13] Singh, S. Aga, and S. Narayanasamy, "Efficiently Enforcing Strong Memory Ordering in GPUs," in Proceedings of the 48th International Symposium on Microarchitecture (MICRO). ACM, 2015.

[14] W. Wilson Jr, "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors," in Proceedings of the 14th International Symposium on Computer Architecture (ISCA). ACM, 1987.

[15] D. A. Wallach, "PHD: A Hierarchical Cache Coherent Protocol," Ph.D. dissertation, Massachusetts Institute of Technology, 1992.

[16] K. Gharachorloo, M. Sharma, S. Steely, and S. Van Doren, "Architecture and Design of AlphaServer GS320," in Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM, 2000.

[17] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous System Coherence for Integrated CPU-GPU Systems," in Proceedings of the 46th International Symposium on Microarchitecture (MICRO). ACM, 2013.

[18] L. E. Olson, M. D. Hill, and D. A. Wood, "Crossing Guard: Mediating Host-Accelerator Coherence Interactions," in Proceedings of the 22th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM, 2017.

[19] J. Alsop, M. D. Sinclair, and S. V. Adve, "Spandex: A Flexible Interface for Efficient Heterogeneous Coherence," in Proceedings of the 45th International Symposium on Computer Architecture (ISCA). IEEE, 2018.

[20] Kumar, A. Kumar, M. Fujita and V. Singh "Validating Multi-Processor Cache Coherence Mechanisms under Diminished Observability" in Proceedings of the 45th International Symposium on Computer Architecture (ISCA). 2019 IEEE 28th Asian Test Symposium (ATS), 2019.

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

[22] Y. Lyu, X. Qin, M. Chen and P. Mishra "Directed Test Generation for Validation of Cache Coherence Protocols" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019.