Int. J. Nonlinear Anal. Appl.
Volume 12, Special Issue, Winter and Spring 2021, 1229-1242
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/IJNAA.2021.5629

Design of MOESI protocol for multicore processors based on FPGA

Raed K. Ibrahim[a], Laith F. Jumma[b], Ibrahim A. Amory[b,*], Aqeel Al-Hilali[b]

[a]*Al-Farahidi University, Baghdad, Iraq.*
[b]*MID, Al-Esraa University College, Baghdad, Iraq.*

(Communicated by Madjid Eshaghi Gordji)

Abstract

Today's multi-core processors are built by all processor manufacturers for computers, cell phones, and other embedded systems. For all computer engineers, designing and researching the hardware architecture of multicore systems is critical. The type of cache coherence protocol employed on a multi-core computer has a direct impact on execution time, latency, and power consumption. Because it is a good example of a CPU, a 32-bit MIPS processor was chosen. With the addition of our prior work, an advanced special circuit was created using VHDL coding and ISE Xilinx software to implement it. One protocol was utilized in this design, the MOESI (Modify, Owned, Exclusive, Shared, and Invalid) protocol. The result of the test was obtained using a test bench, and they revealed that all of the protocols' states were operational.

Keywords: MEOSI protocol, Multicore processor, VHDL, MIPS, FPGA

1. Introduction

Today's CPUs are multi-core processors, with each core having its own cache memory and all being contained on a single chip. The cores share one main memory located outside the chip via a local bus [6]. The architecture of a multi-core system with shared memory is shown in Figure 1. The cache is an important part of the processor design of multi-core computers [28]. The CPU initially looks for data in the cache, and if it isn't found there, it will fetch it from main memory and store it in the cache [24].

*Corresponding author
Email addresses: raed.khalid@alfarahidiuc.edu.iq (Raed K. Ibrahim), laith.f @esraa.edu.iq (Laith F. Jumma), ibrahim.a.amory@esraa.edu.iq (Ibrahim A. Amory), akeel@esraa.edu.iq (Aqeel Al-Hilali)

Received: June 2021 *Accepted:* September 2021

The data could was in another core, and the core could be working in it with changing its value (write operation). As a result, the data in the main memory is corrupted and must be replaced [19]. It is now required that a cache coherence protocol be applied to all caches for all cores [25]. As a result, a spying protocol should use for guarantee the no core uses erroneous data [3].

The aim of this research is to implementation and design of a Cache Coherence controller based on FPGA to achieve best performance by choosing the optimum cache parameters for current application in use. The specific objectives are study the cache coherence mechanism in multi-core domain by means of designing an easy cache and memory to employed as a platform to put into effect the cache coherent protocols. Then focuses in write invalidate type of Snooping Protocols MOESI, with the simulations got the modifications between states of each of the coherence protocols then to evaluate the performance of the proposed new controller unit to managing the cache reconfigurations using hardware description language and implemented on FPGA. Finally witness and analyze the data for educational purpose and to further the development of the different protocols.

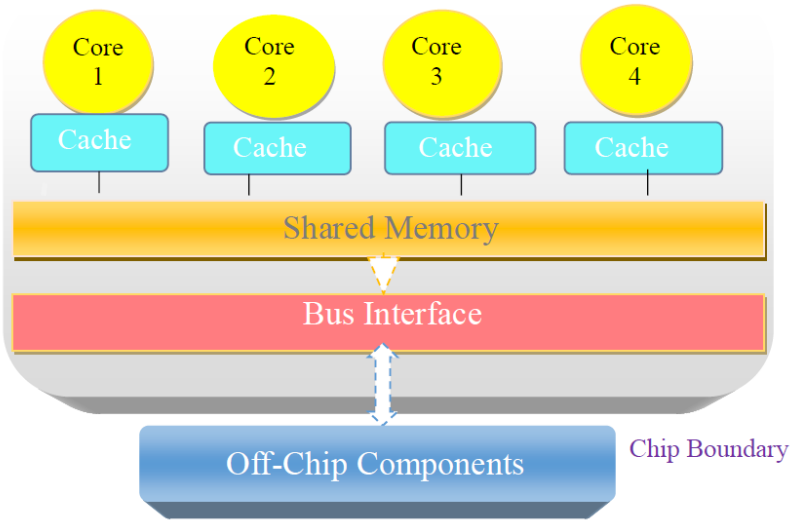


Figure 1: Shared memory multi-core system.

2. Method of research

2.1. Memory cache organization

For cache memory, three different organizing approaches are employed.

Direct Mapping: The Direct mapping function places the incoming block from main memory in specific fixed cache line location. If a number of caches lines (blocks) is L, the incoming block number is B, the cache block number (C) is defined by following equation:

$$C = B \ mod \ L \tag{1}$$

For example, if a cache memory has 8-line size and the incoming memory block number is 22, so the cache block number will equal to (22 mod 8 = 6), then the incoming memory block is placed in line number 6 of the cache memory. Figure 2 explains the organization of some incoming blocks from the main memory into the cache memory [20, 4].

The address issued by CPU was separated in three fields (Tag, Index, and Offset) using direct mapping technique, as illustrated in Figure 2. lengths (in bits) of each field are as follows: If (W) is

the block size in words and (B) is the word size in bytes, the offset field (F) is defined as follows: [5]

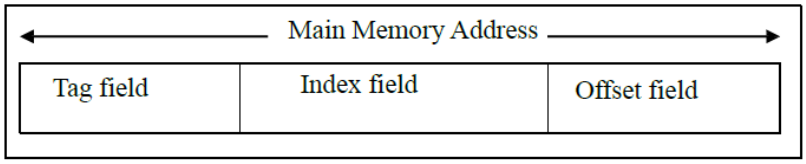
$$F = log_2(w * B) \tag{2}$$

If (L) is the size of the cache in lines, Index field (I) is defined by following equation:

$$I = log_2 L \tag{3}$$

(M) is the number of blocks that make up the main memory's size in bytes, $(M * W * B)$ are the size of main memory in Bytes. So, remaining the higher order bits of main memory address is assigned to the Tag field as the following equation.

$$Tagfield = (log_2(M * W * B)) - F - 1 \tag{4}$$



When the cache memory lines are totally filled and the CPU request for a new memory block, the cache memory line must be replaced by the incoming memory block is determined directly [18].

The direct mapped cache memory protocols started with using the index bits to find the cache memory block that contains a word requested by processor.

The tag comparator looks for a match between the content of the related tag memory and the tag field to be compare to indicate that the corresponding cache memory block which is determined in specific index is currently holding the requested main memory word and cache hit occurs. Among the several words contained with cache block, requested word could to select used an offset field. If no match is found, a cache miss occurs, and a requested block have brought from main memory into cache to make it available to the processor.

The key advantage of direct mapped is the ease with which the cache memory line may be determined directly.

There is one tag comparison needed and that provides faster access time. It is also simple in direct replacement technique.

The main disadvantage of the direct mapped occurs in terms of the possibility of using the same cache memory line for multiple times and this requires frequent replacement of the same cache line while the rest of cache memory lines is not used. For example, if the cache memory size is 32 lines.

Table 1: The difference between cache coherence protocols

MSI protocol	**MESI protocol**	**MOEST protocol**
MSI is basis of three state (Modified(M), shared(S), and Invalid(I))	MSI is basis of four state (Modified(M), Exclusive(E), shared(S), and Invalid(I))	MOEST is basis of five state (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 maim 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

The FPGA (**Field Programmable Gate Array**) is a grid matrix integrated circuit that can be programmed "in the field" without the use of expensive equipment by the user [11]. As shown in figure 2, an FPGA consists of a set of programmable logic gates and extensive interconnect resources from which complex digital circuits can be implemented.

It consists of three main parts:

1. Logic functions are implemented using Configurable Logic Blocks.
2. Programmable Interconnects — these are the ones that do the routing.
3. Programmable I/O Blocks — which allow you to connect to external devices.

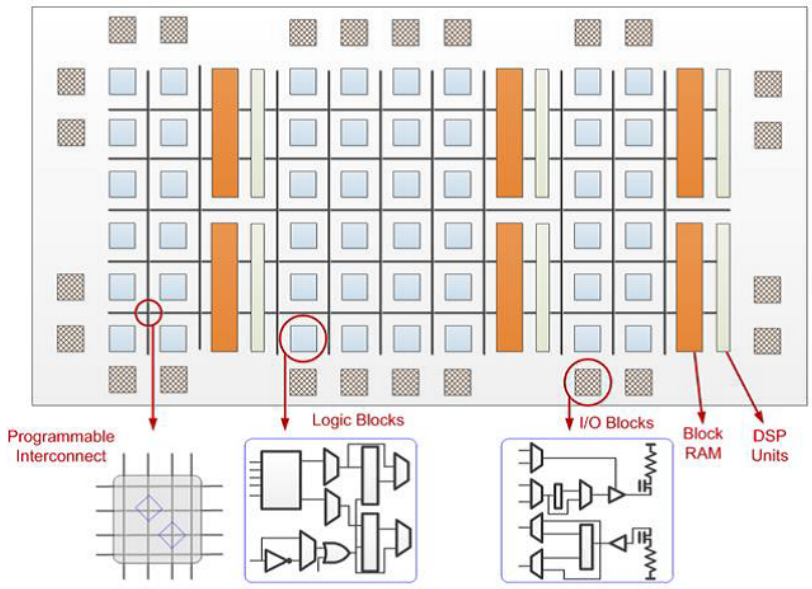


Figure 2: Internal component of FPGA

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 [22].

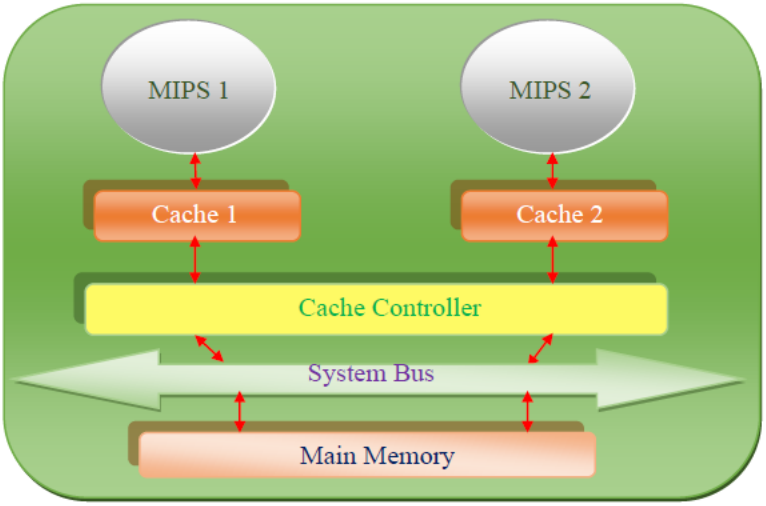


Figure 3: Architecture of dual microprocessor

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 [23].

2.4. Snoopy based coherence protocol

Snoopy Consistency looks for the processor and the cache tag bus at the same time. There are two possibilities for the simultaneous search for cache tags. When the bus receives a priority, the processor snoop is restricted, and when the processor receives a higher priority, the snoop controller cannot respond while the processor is accessing the cache. The solution to the above problem is solved by implementing duplicate labels in the cache. Duplicate tags need to be synchronized here, but tag updates are rare compared to simple searches, reducing precision overhead. Duplicate tags shown in figure 4, [26, 27].

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 MOESI (Modified (M), Owned (O), Exclusive (E), Shared (S) and Invalid (I)). This protocol was developed to MESI exclusive (E), [7, 21].

3. Coherency coherence states

There are different states for each protocol a definition for these states will be given in this section, Table 2 given a definition for each state [17, 2].

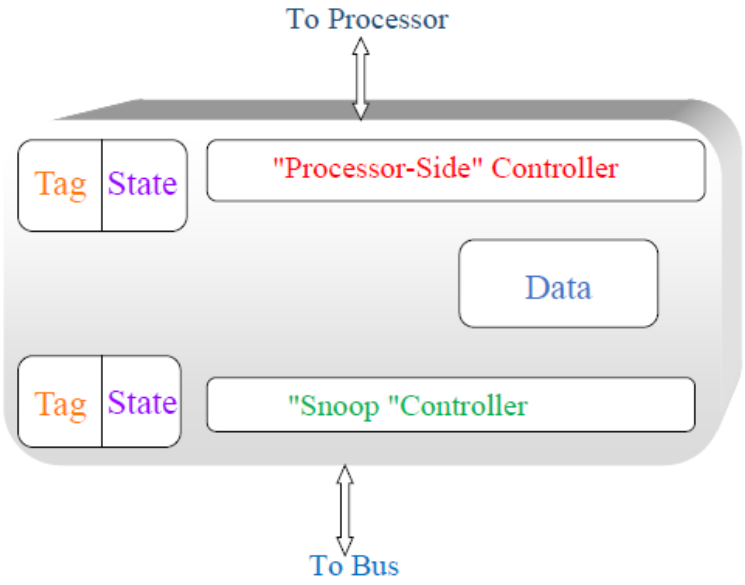


Figure 4: Duplicate tags

Table 2: Definition of cache coherence states

State	Definition
Modified (M)	Only this cache has a valid copy of the line or block. the copy in the main memory is state.
Owned(O)	The block is valid in this cache and another cache but not valid in memory.
Exclusive (E)	The block or line of cache is valid in this cache and main memory.
Shared(S)	The block or line is valid in this cache and at least one more cache and main memory.
Invalid(I)	This state means that the bloch of data is not resident in this cache or it is not useful.

4. MOESI protocol

The MOESI protocol [12] is a more advanced variant of the MESI protocol [10], with a fifth state called Owned (O) that helps decrease memory accesses [9]. Figure 5 depicts the MOESI protocol's state diagram. On multiprocessor systems, there is only one valid data block status cache, and this cache is valid data even if it is shared in main memory to reduce main memory access. You can directly copy the block to another cache. As a result, the state of ownership encompasses both the modified and shared states.

Because the data in the central memory is not updated, dirty values can be shared with other caches, [13]. This signifies that the owner-owned state cache has an accurate data block, and other caches in Memory may also have an accurate data block. Another benefit of this protocol is that it is simple to implement. The transition between MOESI phases is depicted in Table 3.

Table 3: The action of MOESI state

States	Processor Load	Processor Store	Processor Eviction	Incoming Read Reg	Incoming Write Req
Modified	Hit	Hit	Write-back and invalid	Send data & Change to Owned	Send Data and Invalid
Owned	Hit	Change to modified	Write-back and invalid	Send Data, Write-back and Shared	Send Data and Invalid
Shared	Hit	Change to modified	Silent eviction change to invalid	None	change to invalid
Invalid	Change to Exclusive or shared	Change to modified	None	None	None

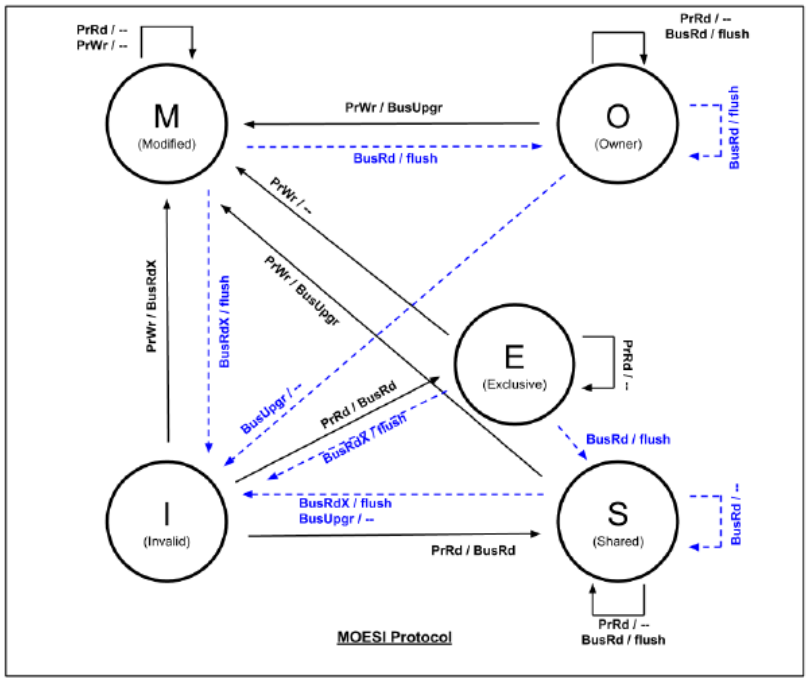


Figure 5: State transition diagram for the MOESI protocol

- **Advantage of MOESI Protocol:**

1. Avoid extra CPU stalls when writing to main memory.
2. If only one cache can be owned (modified), other processors in the same block will be in shared state.

- **Disadvantage of MOESI protocol:**

  In MOESI's Snoopy implementation, when a read request arrives on the bus, every shared in the block sends the requested data. Let the message appear on the bus.

5. Cache coherency controller design

As in our previous work [9], two multi-core MIPS1 and MPIS2 processors were designed, each with its own cache and sharing the same main memory. Using FSM, a cache coherency controller is created that consists of two parts: a coherency tag and a coherence controller (Finite State Machine). The main memory is off chip, and all of these components are connected together on chips. Figure 6 shows a block diagram of the cache coherency controller when it is connected to the two processors on the same chip. There are two caches in the design: cache A and cache B. The caches is a set association that is directly mapped. Each cache has eight sets, with four 32-bit data, a 26-bit tag, and a valid bit, update, dirty, and valid bit in each set. The write and read functions in the cache are performed by two processor components, MIPS1 and MIPS2. The shared memory that was used in the design has 32 entries. The bus controller is used to synchronize access to the shared bus by multiple modules at the same time.

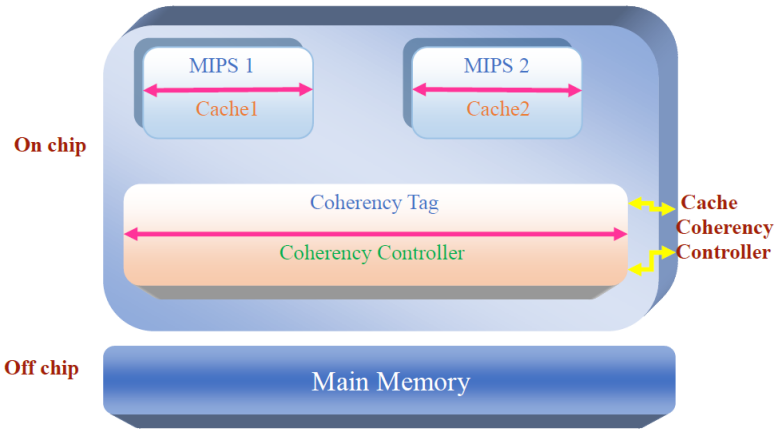


Figure 6: Block diagram for the cache coherency controller

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.

6. Cache coherence controller for MOESI protocol:

Two bits for M (Modify), one bit for O (Owned), two bits for E (Exclusive), one bit for S (Shared), and two bits for I (Insert) were used to indicate different states for the third type of protocol (Invalid). The coherency tag for MOESI states is shown in Figure 7 with 8-bits. MOESI states are displayed in Table 4.

But a new state has been added called Owned state and the rest are the same as in the previous protocols. When Owned (0 or 1); this means this copy has been modified, but there may be other copies in shared state.

Table 5 shows the different sates of MOESI protocols. Only two states will be explained in this section:

1. For (St3) if (M=10, O=0, E=00, S=0, I=00) the OutMOESI1 will be Owned in MIPS1 Hit read (Direct read).

2. For (St9) if (M=01, O=0, E=00, S=0, I=00) the OutMOESI2 will be Not Owned in MIPS2 Hit write (Direct write).

Table 4: MOESI states

M (Modify)	O(Owned)	E(Exclusive)	S(Shared)	I (Invalid)
Modify				
00	Not modified			
01	Modified field MP1			
10	Modified field MP			
Owned				
0	Not owned			
0	Owned			
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 (Invalid1)			
10	Not Valid MP2 (Invalid2)			

Figure 7: Coherency tag with 8-bits for MOESI states

Table 5: State for MOESI protocol

MP1 HTT Read (Direct Read)					
State	M (Modify)	O(Owned)	E(Exclusive)	S(Shared)	I (Invalid)
st0	00	0	00	0	00
OutMOESI1	00	0	01	0	00
st1	00	0	01	0	00
	01	0	00	0	00
	00	0	00	1	00
		1	00	0	00
st2	00	0	10	0	00
OutMOESI1	00	0	00	1	00
st3	10	0	00	0	00
OutMOESI1	00	1	00	0	00
MP1 HTT Write (Direct Write)					
State	M (Modify)	O(Owned)	E(Exclusive)	S(Shared)	I (Invalid)
st8	00	0	00	0	00
	00	0	01	0	00
	00	0	10	0	00
	00	0	00	1	00
	01	0	00	0	00
	10	0	00	0	00
	00	1	00	0	00
OutMOESI1	01	0	00	0	00
MP1 MISS Read (Direct Read)					
State	M (Modify)	O(Owned)	E(Exclusive)	S(Shared)	I (Invalid)
st 10	00	0	01	0	00
MP1 MISS Write (Direct Write)					
State	M (Modify)	O(Owned)	E(Exclusive)	S(Shared)	I (Invalid)
st 11	01	0	00	0	00
st 12	00	0	10	0	00
st 13	10	0	00	0	00
MP2 HTT Read (Direct Read)					
State	M (Modify)	O(Owned)	E(Exclusive)	S(Shared)	I (Invalid)
st4	00	0	00	0	00
OutMOESI2	00	0	10	0	00
st5	00	0	10	0	00
	10	0	00	0	00
	00	0	00	1	00
	00	1	00	0	00
st6	01	0	00	0	00
OutMOESI2	00	0	00	0	00
st7	00	0	00	0	00
OutMOESI2	00	0	00	1	00

MP2 HTT Write (Direct Write)					
State	M (Modify)	O(Owned)	E(Exclusive)	S(Shared)	I (Invalid)
st9	00	0	00	0	00
	00	0	01	0	00
	00	0	10	0	00
	00	0	00	1	00
	10	0	00	0	00
	01	0	00	0	00
	00	1	00	0	00
OutMOESI2	10	0	00	0	00
MP2 MISS Write (Direct Write)					
st400	Do NOTHING				

7. Simulation results of the MOESI protocol

Two bits for M (Modify), one bit for O (Owned), two bits for E (Exclusive), one bit for S (Shared), and two bits for I (Include) were used to denote distinct states for the third type of protocol (Invalid).

But a new state has been added called Owned state and the rest are the same as in the previous protocols. When Owned (0 or 1); this means this copy has been modified, but there may be other copies in shared state.

Only two states will be explained in this section:

1. For (St3) if (M=10, O=0, E=00, S=0, I=00) the OutMOESI1 will be Owned in MIPS1 Hit read (Direct read). Figure 8 shows the test bench of MOESI protocol-MP1 Hit Read (Direct Read).

Figure 8: Test bench of MOESI protocol-MP1 Hit Read (Direct Read)

2. For (St9) if (M=01, O=0, E=00, S=0, I=00) the OutMOESI2 will be Not Owned in MIPS2 Hit write (Direct write). Figure 9 shows the test bench of MOESI protocol-MP2 Hit Write (Direct Write).

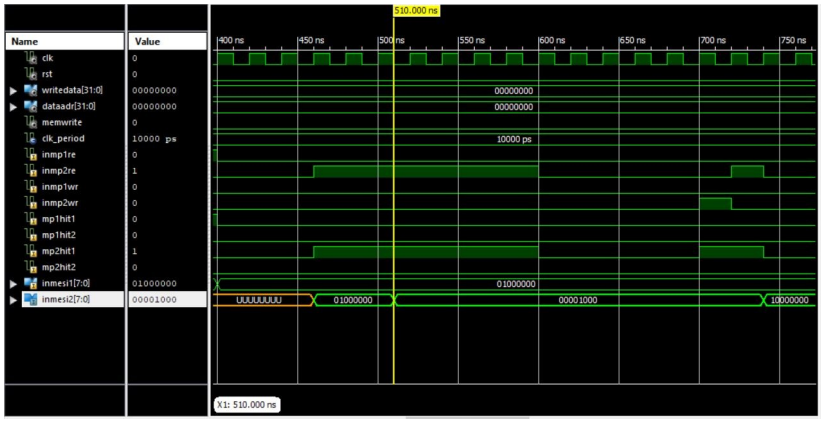


Figure 9: Test bench of MOESI protocol-MP2 Hit Write (Direct Write)

8. 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 6: State for MOESI protocol

Devise Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of slice Registers	1266	54576	2 %
Number of slice LUTS	16814	27288	61 %
Number of fully used LUTS-FF pairs	1233	16847	7 %
Number of bonded IOBS	68	296	22 %
Number of BUFG/BUFGCTRLs	8	16	50 %
Number of DSP48A1s	16	58	27 %

9. Conclusion

The Cache Coherency Controller was created in anticipation of our earlier work [9] for MOESI protocols. A unique circuit is built using VHDL coding and implemented using ISE Xilinx software in this work. The results of the tests were obtained utilizing a test bench, and they revealed that all of the protocol's states were operational.

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. For future studies, the current work can be expanded to include the internet of things, cloud computing, and possibly e-government. Additionally, ZigBee technology has the potential to augment the current work [14]-[16].

References

[1] T. Abd, Y. S. Mezaal, M. S. Shareef, S. K. Khaleel, H. H. Madhi, and S. F. Abdulkareem. *Iraqi e-government and cloud computing development based on unified citizen identification*, Periodicals of Engineering and Natural Sciences, 7 (4) (2019)1776-1793.

[2] 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.

[3] 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.

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

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

[6] R. Fuchsen, *How to address certification for multicore based IMA platforms: current status and potential solutions*, Digital Avionics Systems Conference (DASC), 3 (12) (2010).

[7] 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.

[8] Z.K. Hussein, H.J. Hadi, M.R. Abdul-Mutaleb, Y.S. Mezaal, *Low cost smart weather station using Arduino and ZigBee*, Telkomnika , 18 (1) (2020)282-288.

[9] A. Ibrahim, H. Ahmed and F. Jumma, *MESI Protocol for Multicore Processors Based on FPGA*, in Periodicals of Engineering and Natural Sciences, 7 (2) (2019)10-13.

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

[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] B. 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.

[13] 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.

[14] A.A.H. Mohamad, Y. S. Mezaal and S. F. Abdulkareem, *Computerized power transformer monitoring based on internet of things*, International Journal of Engineering & Technology 7, (4) (2018) 2773-2778.

[15] Ch. Meshram, W. Ibrahim, J. Obaid, S. Gajbhiye Meshram, A. Meshram and A. Mohamed Abd *El-Latif, Fractional chaotic maps based short signature scheme under human-centered IoT environments*, Journal of Advanced Research, 2020, ISSN 2090-1232, https://doi.org/10.1016/j.jare.2020.08.015. .

[16] A.J. Obaid , *Critical Research on the Novel Progressive, JOKER an Opportunistic Routing Protocol Technology for Enhancing the Network Performance for Multimedia Communications*, 1254 (2021). https://doi.org/10.1007/978-981-15-7527-3_36.

[17] 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.

[18] 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, 1984.

[19] F. Pong and M. Dubois, *A new approach for the verification of cache coherence protocols*, IEEE Trans. Parallel Distrib. Syst., 6 (8) (1995) 773–787.

[20] 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.

[21] 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.

[22] 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.

[23] A. 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.

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

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

[26] A. W. Wilson Jr, *Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors*, in Proceedings of the 14th International Symposium on Computer Architecture (ISCA). ACM, 1987.
[27] D. A. Wallach, *PHD: A Hierarchical Cache Coherent Protocol*, Ph.D. dissertation, Massachusetts Institute of Technology, 1992.
[28] P. S. a. W. Zhang, *WCET Estimation of Multi-Core Processors with the MSI Cache*, 2013.