International Journal of Engineering Science Invention (IJESI)
ISSN (Online): 2319-6734, ISSN (Print): 2319-6726
www.ijesi.org ||Volume 9 Issue 5 Series I || May 2020 || PP 44-55

MSI Protocol for Multicore Processors Based on FPGA

Mays K. Faeq[1], Safaa S. Omran[2].
1,2 (Electrical Engineering Technical College, Middle Technical University, Iraq)

ABSTRACT:*Memory hierarchy design plays an important role in improving the performance of chip multiprocessor (CMP).The reason is that the performance of a CMP is strongly affected by the latency of fetching data from the memory system. Several organizations of the memory hierarchy have been explored to optimize this latency. In the memory hierarchy, data traversing is based on a cache coherence protocol which is the skeleton of the CMP's memory system. Multicore need for a special protocol for the purpose of loading block or line of data from memory to cache. The MSI protocol was used to build a Cache controller circuit that performs the process of controlling the cache contents of each processor as well as the memory. The MIPS type processor was used with the Direct Mapped (D.M) cache memory. This circuit was implemented for educational purposes in advanced computer material for graduate students and the practical results were consistent with the theory.*

KEYWORDS -*Cache Coherency, Multicore Processor, MSI protocol, MIPS Processor, FPGA, VHDL.*

Date of Submission: 06-05-2020 Date of Acceptance: 20-05-2020

I. INTRODUCTION

Nowadays, chip multiprocessor (CMP) 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. Moreover, modern technologies support millions of transistors to be integrated in one chip which eases the design of multicore on chip in terms of area. In fact, several CMPS have been commercialized in the market [1]- [4]. Therefore, architects have moved towards multi core architectures as an effective way of achieving higher performance. All commercial designs that have been announced since 2002 have had a moderate number of cores with shared memory architectures maintained with snoopy cache coherence protocols [5], [6]. 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 [7]. Today’s processors solve the cache coherence problem in hardware by implementing a cache coherence protocol. These cache coherence protocols are based on the idea that each cache snoops or watches all activity on a global bus, or is informed about such activity by some global broadcast mechanism. Appropriate coherence actions should be taken based upon what is detected on the bus. There are two primary methods to maintain the coherence: write invalidate and write update. In write invalidate all other cache copies are invalidated. The other method is writing update in which the data item is written and updated by all caches. All cache coherence protocols use one of the two schemes. The protocols differ in how they respond to the different events that are snooped on the global bus [8]-[11]. In a shared memory multiprocessors system with a separate cache memory for each processor, it is possible to have many copies of the operand, such as one copy might be residing in the main memory while other copies reside in the cache memory. When one copy of an operand changes then the other copies should also be updated. Thus, cache coherence ensures that changes in all operands are propagated throughout the system in a timely fashion [12]. Cache coherence is the consistency of shared and stored the data in multiple local caches [13].

Cache coherence protocols are therefore employed to solve the problem of inconsistent data that may arise from updating or writing on the same memory location by different multiprocessor cache. Moreover, these protocols help to enforce the desired memory model and guarantee that access to any memory location still returns the latest value written in that location. Consequently, cache coherence protocols can be considered as the solution for maintaining data consistency between the shared cache blocks [14]. Maintaining cache coherence could be achieved via either software or hardware solution. In both approaches, updates made by one processor should be communicated to others and there should be a consistent view of the memory of all processors. So, the main idea is that the processor should broadcast all its operations to all other processors, and all other processors should change the state of shared block accordingly [14]. Multicore chips are entering commercial and consumer markets vastly. They began in special-purpose, niche markets such as high-performance graphics and networking. Recently, multicore chips have been introduced into the general-purpose market as well. A multi-core processor is an integrated circuit (IC) has two or more processors, have been attached in single chip to enhance the performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks [15]. Multicore processor needs parallel software to achieve continuing performance gains. Most parallel software in the commercial market relies on the shared-memory programming model in which all processors access the same physical address space [16]. In multicore processor that had a shared memory system, a single shared address space maybe read or write by each processor core. The memory consistency model defines the architecturally visible behavior of its memory system in a shared memory machine. It provides rule about stores and loads and how they are doing upon the memory. As part of memory consistency model, many machines provide cache coherence protocols which guarantee that multiple copies of data in caches are kept up-to-date. The main job of consistency is to define the correctness of shared memory [17]. Figure 1 shows the dual multicore processor.

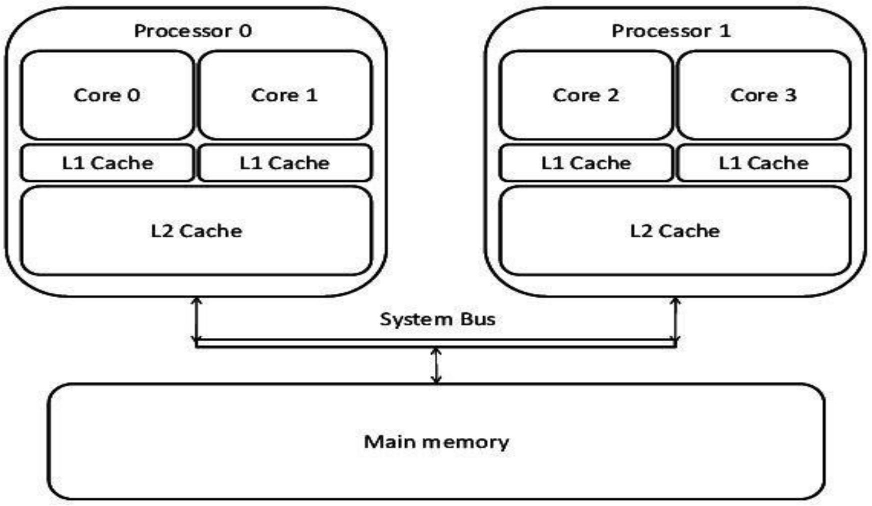

Fig 1. Dual multicore processor.

II. RESEARCH METHOD

A. THE CACHE

A cache is a small size and high-speed memory that stores data temporarily from some frequently used addresses which is in the main memory. It is used in modern CPUs to shorten the time cycle required by the processor to fetch the requested data thus effectively improving the CPU performance [18]. Cache memory is being used in the modern computer systems, to temporarily hold the currently used contents of the main memory locations. Data present in the cache memory can be accessed in less time than from main memory; therefore, cache memory is faster memory. Cache memory is expensive than main memory even though it is much smaller than main memory [19]. Data is transferred between memory and cache in blocks of fixed size, called cache lines. A cache entry is created, when a cache line is copied from memory to the cache. The cache Tags are added to the cache entry along with data in order to supply the remaining bits of address, which are used to differentiate memory locations that are mapped to the same cache block. When the processor wants to read from a location or write to a location, it first checks for a corresponding entry in the cache. If the location is found in the cache then cache hit has occurred. If cache hit occurs the processor immediately reads the data from the cache line or writes the data into the cache line. If the location is not found in the cache, then cache miss has occurred, so it reads the data from main memory and copies this data into the cache [20].

B. ARCHITECTURE DIRECT MAPPED CACHE

A direct mapped cache has one block in each set. The first S memory addresses are directly mapped onto the S cache blocks, and then the mapping wraps around, and the S next memory addresses also maps onto the S cache blocks. As there are many memory addresses mapped to a same set, the actual address of the data contained in each set must be tracked. The least significant bits (LSB) of the address are used to specify the cache set that holds the data. The remaining bits, called the tag, are stored to indicate the actual memory address. In this cache organization, each location in main memory can go in only one entry in the cache. One way set associative cache is the other name for direct mapped cache [21],[22]. Figure.2 shows a simple direct-mapped cache with four-word block size purposely designed for this project. The constructed cache has 8 sets, each consist of four 32-bits of data, 26-bits of tag, and 1 valid bit. The cache is accessed using the 32-bits address which consists of bits 0 to 1for byte *select*, bits 2 to 3for *word select*, bits 4 to 5 for index, and the rest bits for the *tag*. The *byte select* is used as function of the cache for it to be able to read variable data for example - CPU requesting only 1- byte or 2-byte data and the *word offset* on the other hand, used to select any words inside the cache. The *tag* bit is for representing the tag address requested by the CPU and to be compared to the tag addresses inside the cache [23]. Figure 3. Show the design of Direct mapped cache.

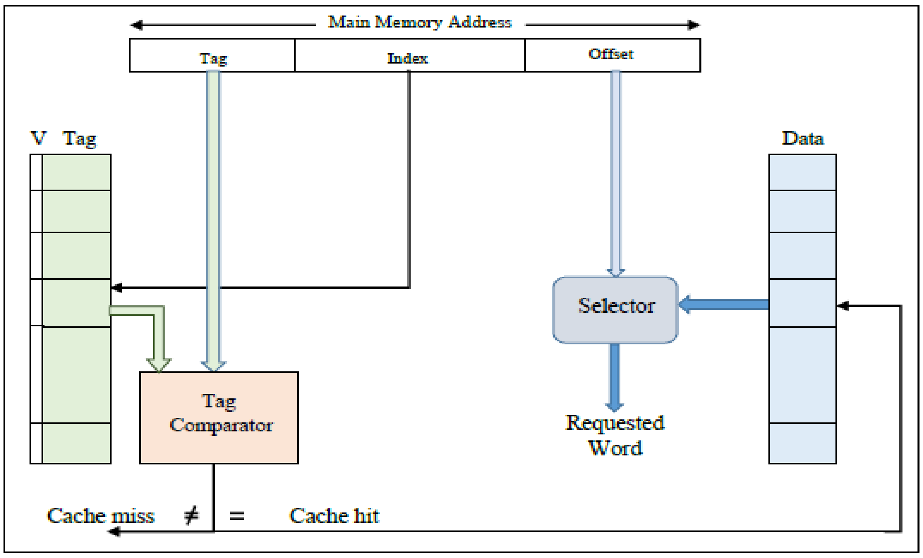

Fig 2. Direct mapped cache protocols

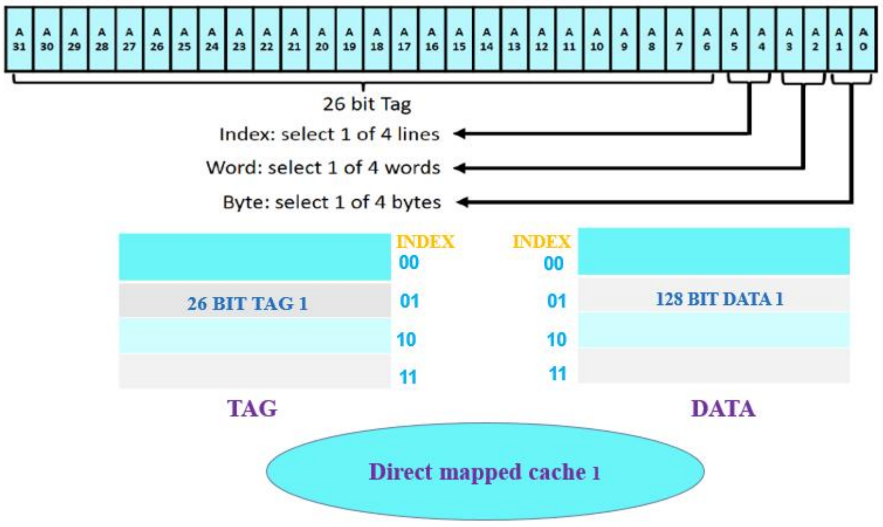

Fig 3. Direct mapped cache design.

C. MULTI-CORE MIPS PROCESSOR DESIGN

Two of single core MIPS processor are combined together to generate a multicore MIPS processor. Each single core is a pipelined MIPS processor and has its own L1 cache memory. Both cores shared the main memory. Multicore processor exploits the parallel available in program to allow its cores to work together for the same job, therefor parallel program is needed to reach better performance from multicore processor. Since each core has L1 cache, then the same memory address may be found in both cores. This may cause consistency problems in data cache. Instruction cache does not have this problem because the processor cannot modify program instructions. In this paper snooping based coherency is used as a cache coherency mechanism, and the coherency protocol used is MSI protocol.

D. CACHE COHERENCY MECHANISM

To implement cache coherency protocols in the Multicore system and manage the consistency of memory, cache coherency mechanism is used. Snooping based coherency is used as a cache coherency mechanism. It allows each cache to monitor the address lines so that to gain access to main memory which they may have a copy of its data. Any activity on cache line will trigger message, which will be broadcasted to all the caches to update the cache line with the activity.

E. SNOOPY BASED COHERENCE PROTOCOL

In snoopy coherence both processor and bus lookup for cache tag at the same time, as shown in figure 4 the Simultaneous Cache Tag Lookup There may be two possibilities; if bus receives priority then processor snoop is restricted and if processor receives high priority then snoop controller cannot respond during processor accessing cache. Solution to the above problem is implementing duplicate tags in cache as shown in figure 5here duplicate tags need to be synchronized. But as updating tag is infrequent compare to simple lookup so less overhead for correctness [24].

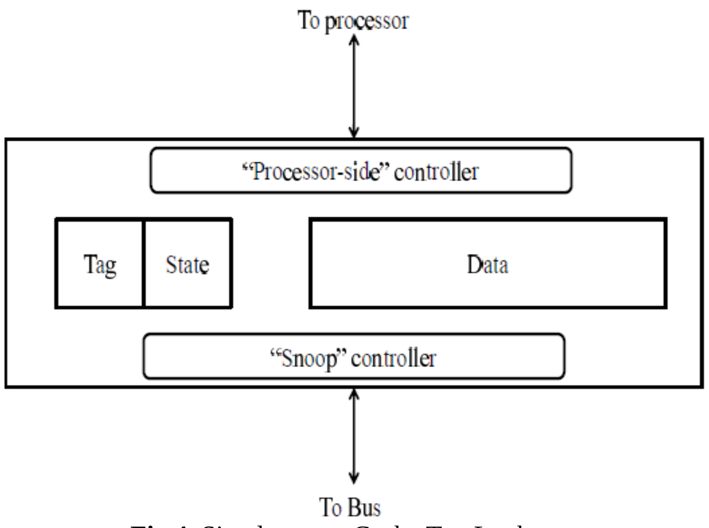

Fig 4. Simultaneous Cache Tag Lookup

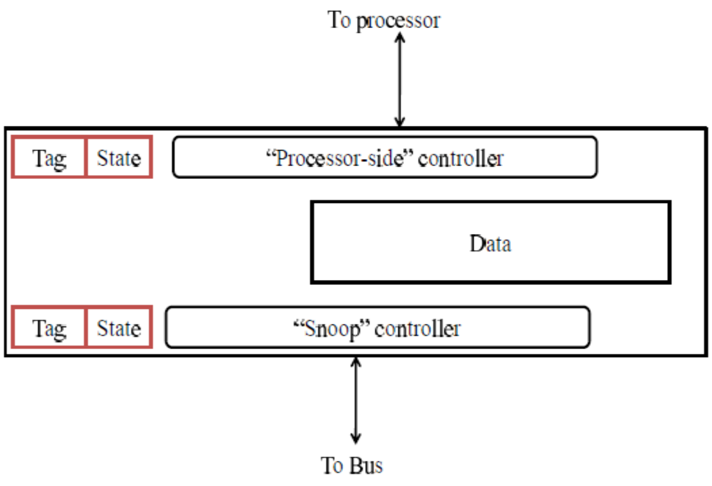

Fig 5. Duplicate tags

III. MSI PROTOCOL

The letters of MSI protocol identifies the states in which a cache line can be "M" *(Modified)* which implies that the block in the cache has been modified [25]. [26],[27], and the data in the cache is different from that of the backing store. The cache with a block in the modified state is responsible for updating the backing store and reading the data but without sending anything out. However, the "S" letter in the MSI means *Shared*. Shared means that the block is unmodified and it is in a read-only state at least in one cache. The cache can get the data back without initially updating the backing store. The last the letter "I" refers to the *Invalid* state which implies that the block is invalid as the data may not present in the current cache or has been invalidated by a bus request, and this block must be fetched from a memory. States' *S*, *M* and *I* are maintained by the communication between the cache and the backing store.

When there is a write to a block which is in the invalid state, the block must be moved to a modified state. A write request must then be sent via the bus. Sending a write request to the bus helps to ensure that no other processor has a copy of this block, so it will be invalidated to all other processors. However, if the block is in the Modified state and there is a write request on the bus, in this case, we have to snoop a write operation on the bus and then give a write back to the Invalid state. In the MSI protocol, if a cache block is in the Modified state and its processor initiates a Bus Read to that block, a Write Back would occur. The cache would then be moved to the Shared state [28]. Figure 6 shows the state transition diagram of MSI protocol.

In MSI protocol Bus-side requests include:

BusRd: BusRd transaction is generated by a PrRd that misses in the cache, and the processor expects a data response as a result. The cache controller puts the address on the bus and asks for a copy that it does not intend to modify. The memory supplies the data.
BusRdX: BusRdX transaction is generated by a PrWr to a block that is either not in the cache or is in the cache but is not in modified state. The cache controller puts the address on the bus and asks for an exclusive copy that it intends to modify. The memory system provides the data. All other caches are invalidated. Once the cache obtains the exclusive copy, the write can be performed in the cache.
Flush: Flush is a snooped request that indicates that an entire cache block is written back to the main memory by another processor.

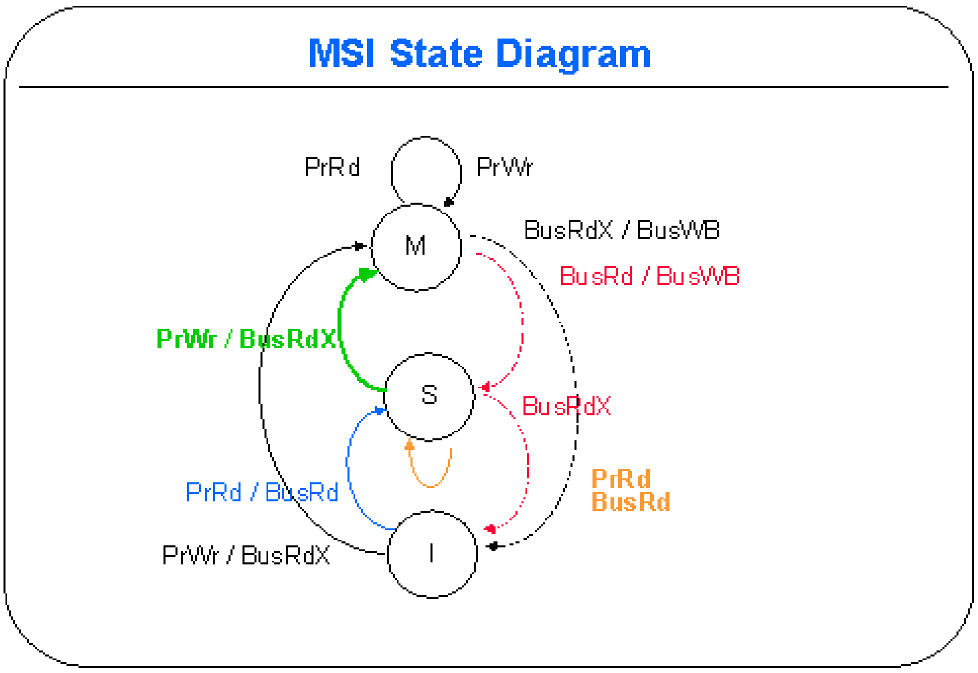

Fig 6. state transition diagram of MSI protocol

The three states in the MSI protocol are used to directly enforce the cache coherence by allowing only a single processor in the modified state at a given time, allowing multiple processors in the shared state concurrently, and disallowing other processors in the shared state while a processor is in the modified state. In the MSI system, there is a serious drawback that an explicit BusRdX transaction is required for a read followed by a write, even if there are no other shared copies. When a processor reads in and modifies a data item, two bus transactions are generated in this protocol even in the absence of shared copies. The first is a BusRd that gets the cache block to S state, and the second is a BusRdX (or BusUpgr) to invalidate other cached copies. The BusRdX is useless in case there are no other shared copies. Table.1. shows the MSI state transitions.

Table 1. MSI state transitions

	PROCESSOR ACTIONS			REMOTE REQUESTS	
STATE	READ	WRITE	EVICTION	READ	WRITE
M	HIT	HIT	WRITE BACK $\rightarrow$ I	SEND DATA $\rightarrow$ S	SEND DATA $\rightarrow$ I
S	HIT	WRITE REQ $\rightarrow$ M	SILENT DROP $\rightarrow$ I	ACK	ACK $\rightarrow$ I
I	READ REQ $\rightarrow$ S	WRITE REQ $\rightarrow$ M	NONE	NONE	NONE

❖ Advantage of MSI protocol:
1. Distinction between modified and shared state.
2. Multiple copies of block can be present at the same time.
3. Shared to Modify transition can be made without reading data from cache.

❖ Disadvantage of MSI protocol:
1.On read, block goes to "S" state, even if it is the only copy present. This is problem because when write request arrive state changes from "S" to "M" and invalidate message is broadcast on the bus, even though it is the only copy present.
2. Unnecessary broadcast of invalidation message.

IV. SYSTEM DESIGN

A VHDL components of MIPS processor was designed [29] is combined with the VHDL components of this design, by using *(Xilinx ISE Design Suite 14.1)* all these components are connected together in order to compose the top level, later a test bench is written and used to enter the 2 bits of cache size controller and execute a test program[30] Figure 7 shows Top_leveland Figure 8 Shows the schematic view of top_level components. It consists of three parts: pipelined MIPS, data memory system and instruction memory system.

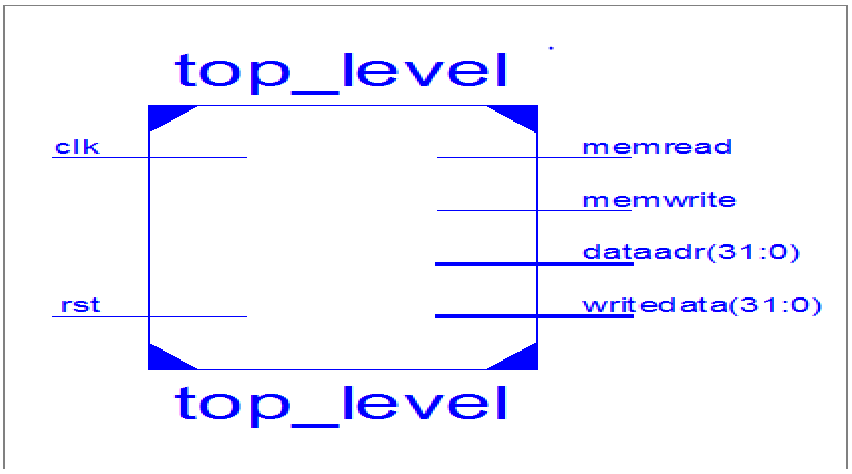

Fig 7. Top_level.

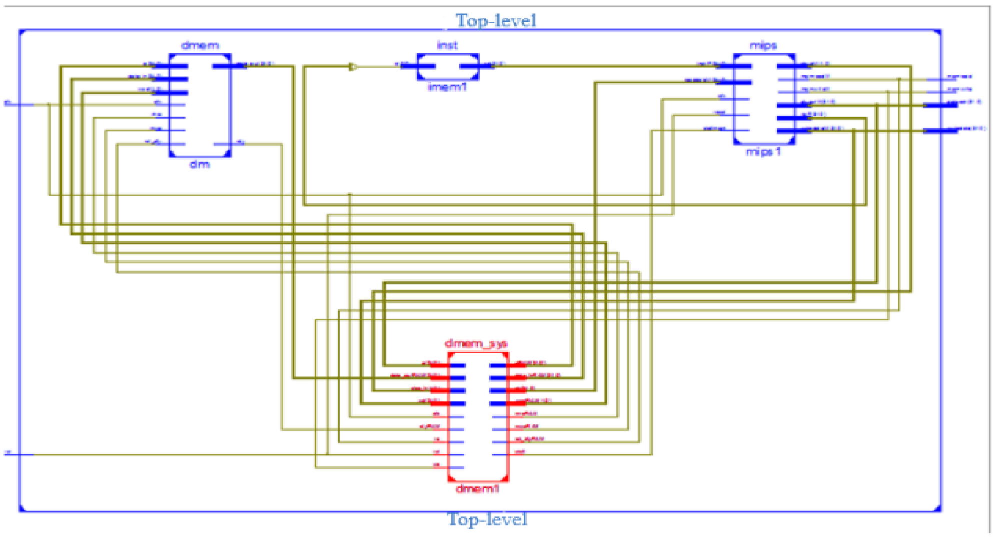

Fig 8. schematic view of top-level components

V. CACHE DESGINE

In this work two microprocessors MIPS1 and MPIS2 were designed and each with separate cache. A cache coherency controller is designed which consist of two parts, the coherency tag and coherence controller by using (FSM). All these components are connected together on chips and have the main memory off chip. Figure 9 shows the design cache coherency protocol. Figure 10 shows the RTL Coherence tag and coherence controller**.**

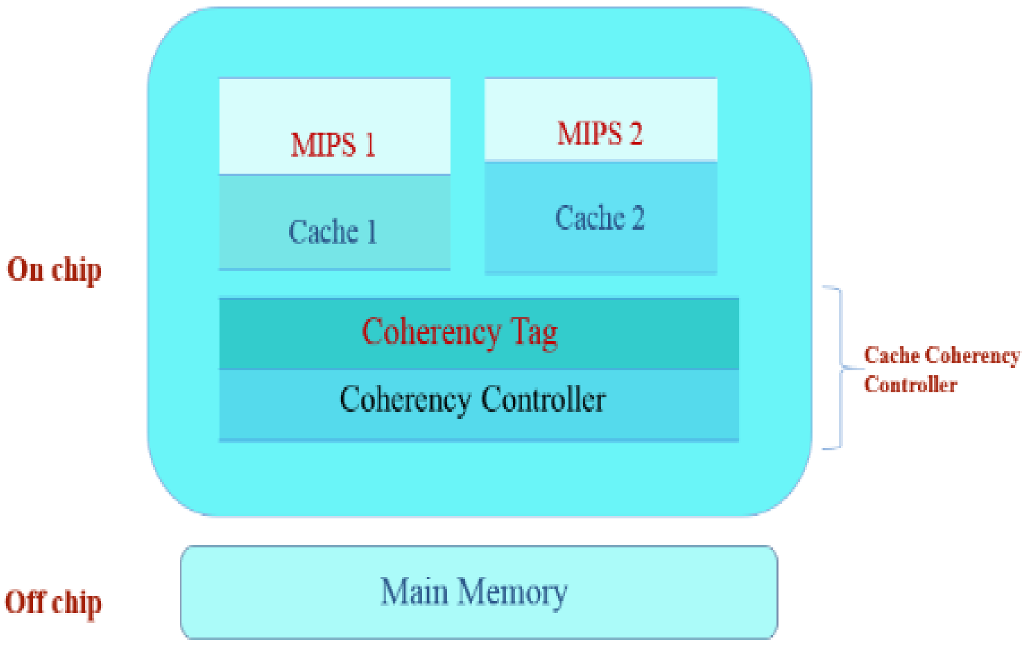

Fig 9. Design of cache coherency protocol.

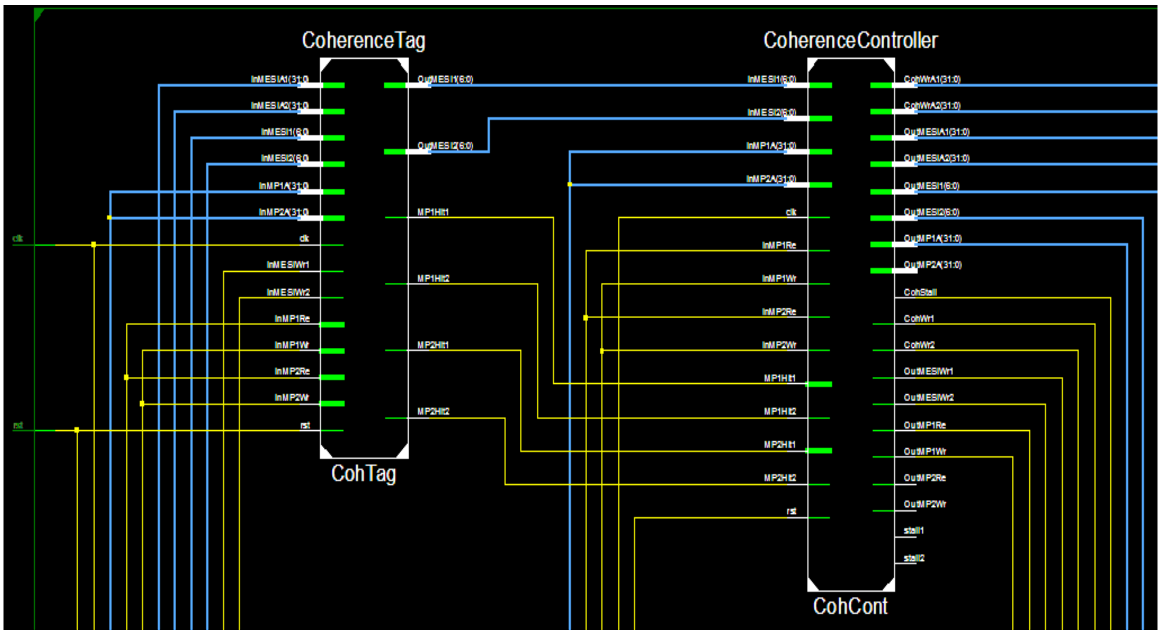

Fig 10. RTL Coherence tag and coherence controller

In this paper a 5- bits where used to indicate different states for MSI protocol, two bits for M(*Modify*), one bit for S(*Shared*) and two bits for I(*Invalid*). Table 2 shows MSI states. Figure 11 shows the coherency tag with 5-bits for MSI states and table 3 shows the different sates of MSI Protocol.

Table 2. MSI states.

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

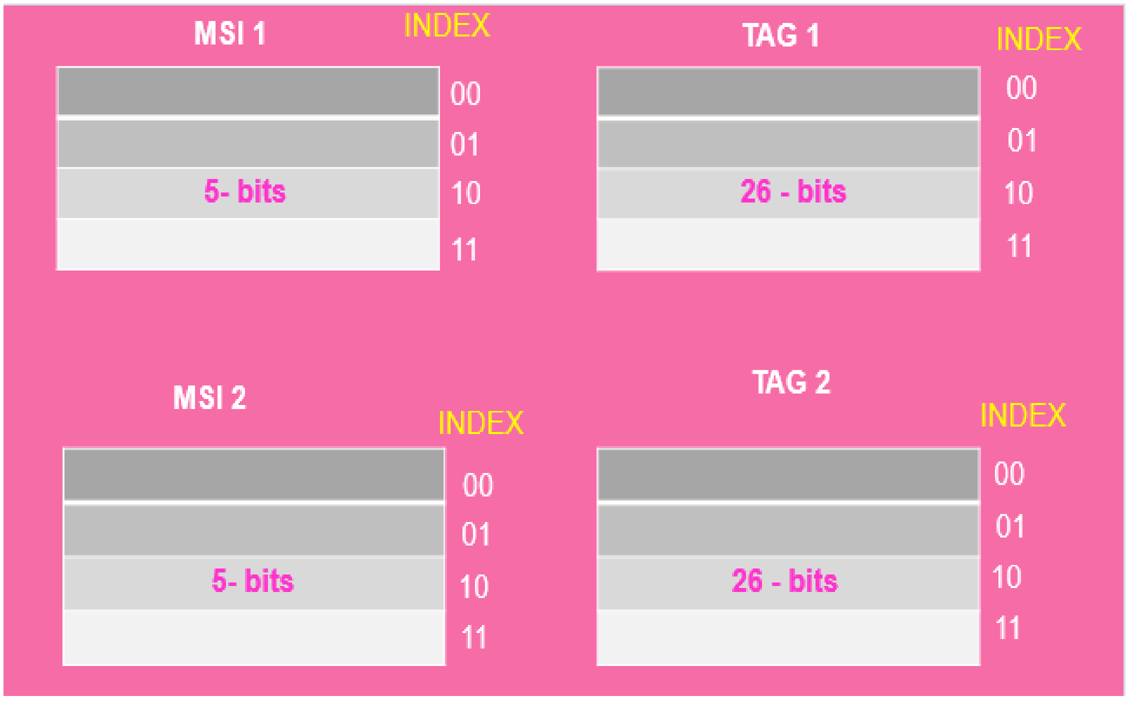

Fig 11. Coherency tag with 5-bits MSI states.

Table 3. Sates of MSI protocol.

MP1 HIT Read (Direct Read)			
State	M *(Modify)*	S *(Shared)*	I *(Invalid)*
St0 Out MSI1	00 00	0 0	00 01
St1 Nothing changes	00 01 00	0 0 1	01 00 00
St2 Out MSI1	00 00	0 1	10 00
St3 Out MSI1	10 00	0 0	00 10
MP1 HIT Write(Direct Write)			
State	M *(Modify)*	S *(Shared)*	I *(Invalid)*
St8 Out MSI1	00 00 00 00 01 01	0 0 0 1 0 0	00 01 10 00 00 00
St9 Out MSI1	10 01	0 0	00 00
MP1 MISS Read(Direct Read)			
State	M *(Modify)*	S *(Shared)*	I *(Invalid)*
St12	00	0	01
MP1 MISS Write(Direct write)			
State	M *(Modify)*	S *(Shared)*	I *(Invalid)*
St13	01	0	00
MP2 HIT Read(Direct Read)			
State	M *(Modify)*	S *(Shared)*	I *(Invalid)*
St4 Out MSI2	00 00	0 0	00 10
St5	00 10 00	0 0 1	10 00 00
St6 Out MSI2	00 00	0 1	01 00
St7 Out MSI2	01 00	0 0	00 10
MP2 HIT Write(Direct Write)			

State	M (Modify)	S (Shared)	I (Invalid)
St10	00	0	00
	00	0	01
	00	0	10
	00	1	00
	10	0	00
Out MSI2	10	0	00
St11	01	0	00
Out MSI2	10	0	00
MP2 MISS Read(Direct Read)			
State	M (Modify)	S (Shared)	I (Invalid)
St14	00	0	10
MP2 MISS Write (Direct write)			
State	M (Modify)	S (Shared)	I (Invalid)
St15	10	0	00
St400	DO NOTHING		

VI. RTL SCHEMATIC OF MULTI-CORE MIPS PROCESSOR DESIGN

Two single core MIPS processors are combined together to generate a multicore MIPS processor. Each single core is a pipelined MIPS processor and has its own L1 cache memory. Both cores shared the main memory. Multicore processor exploits the parallel available in program to allow its cores to work together for the same job, therefor parallel program is needed to reach better performance from multicore processor. Since each core has L1 cache, then the same memory address may be found in both cores. This may cause consistency problems in data cache. Instruction cache does not have this problem because the processor cannot modify program instructions, in this paper snooping-based coherency is used as a cache coherency mechanism, and the coherency protocol used is MSI protocol, Cache controller is used to regulate cache memory. When a processor wants to access memory location, it first sends address to cache 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. The design implemented using VHDL (Very high speed integrated circuitHardware Description Language) then integrated with FPGA (Field Programmable Gate Arrays) Xilinx Spartan 6. The results for the different parts of the processor are presented in the form of test bench waveform and the architecture of the system is demonstrated and the result was matched with theoretical result. Figure 12 shows the Cache coherency controller for MSI protocol.

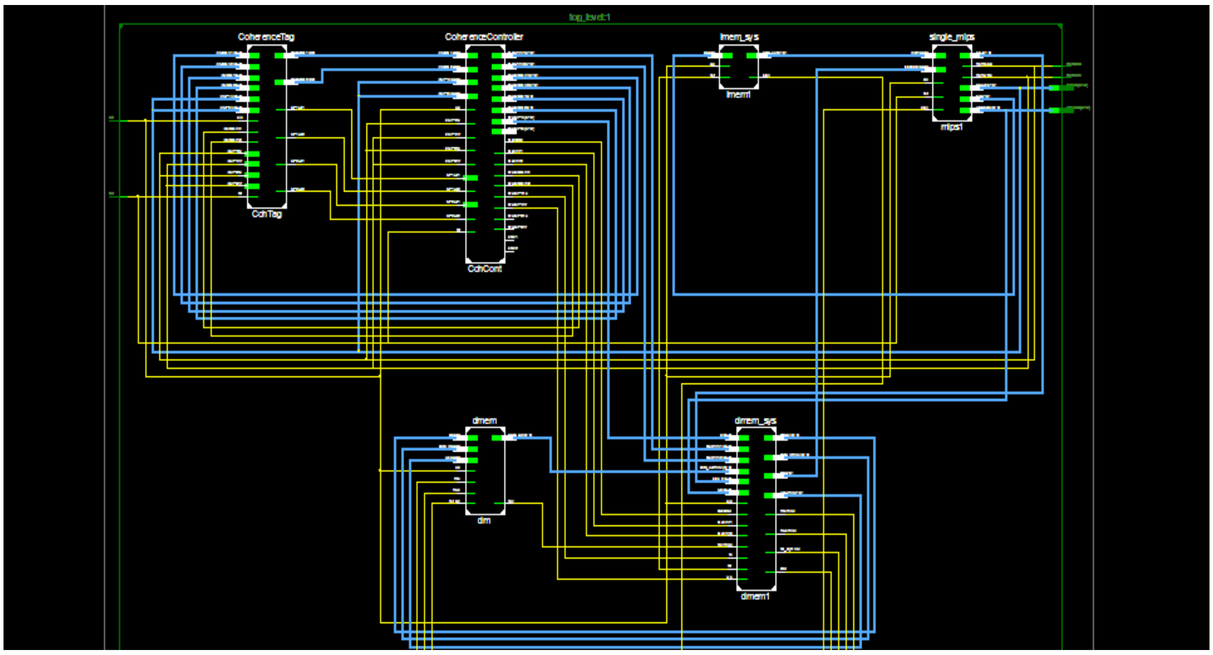

Fig 12. Cache coherency controller for MSI protocol

VII. RESULTS AND DISCUSSIONS

To verify the validity of the design that was built in this paper, multiple programs where written for the purpose of examining the work of the MSI protocol by the coherency controller. The results were found to be identical to the different 15 states that designed in the protocol, Figure 13 shows the test bench results for some of these states.

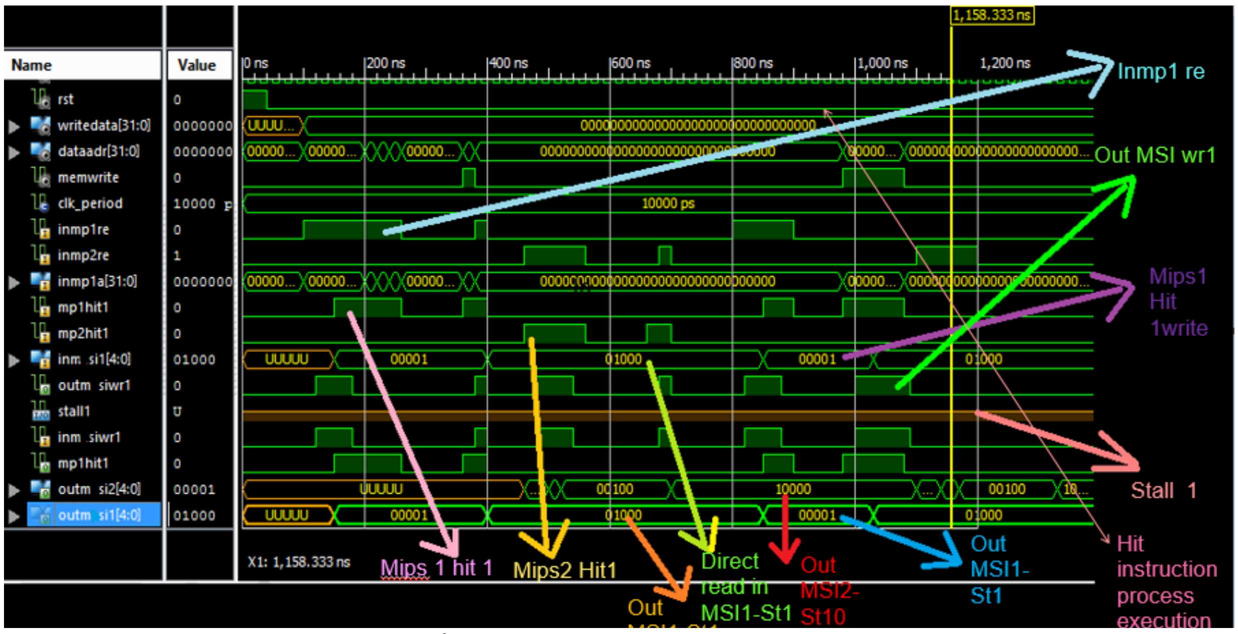

Fig 13. Test Program Execution

VIII. CONCLUSIONS

In this paper, a pre-designed mips type processor was used in the department where I work at the college. Using this processor to build another processor corresponding to it so that we 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 MSI protocol, and the results were found to be compatible with the topic design and the purpose of this research for use in the scientific purposes of master's students in advanced computer technique**.**

REFERENCES

[1]. Chen, Thomas, et al., "Cell broadband engine architecture and its first implementation - a performance view," IBM Journal of Research and Development 51.5 (2007): 559-572.

[2]. George, Varghese, T. Piazza, and H. Jiang., "Technology Insight: Intel Next Generation Microarchitecture Codename Ivy Bridge," (2011).

[3]. Conway, Pat, et al., "Cache hierarchy and memory subsystem of the AMD Opteron processor," IEEE micro 30.2 (2010): 16- 29.

[4]. P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32- way multi- threaded spark processor," IEEE Micro, vol. 25, pp. 21-29, March 2005.

[5]. Tendler, J.M.; Dodson, J.S.; Fields, J.S.; Le, H.; Sinharoy, B.: POWER4 system microarchitecture. IBM J. Res. Dev. 46(1), 5– 25 (2002).

[6]. Datasheet, Intel Pentium D Processor. Intel (2006).

[7]. Veenstra, J.; Fowler, R.: A performance evaluation of optimal hybrid cache. In: Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (1992).

[8]. Radhakrishnan, S.; Chinthamani, S.; Cheng, K.: The blackford northbridge chipset for the Intel 5000. Micro IEEE 27(2), 22–33 (2007).

[9]. Cheng, L.: Context-Aware Coherence Protocols for Future Processors. Ph. D Thesis. University of Utah (2007).

[10]. AMD, AMD64 Architecture Programmer's Manual Vol 2 'System Programming. AMD.

[11]. Solihin, Y.: Fundamentals of Parallel Computer Architecture Multichip and Multicore Systems. Solihin Publishing & Consulting LLC (2008).

[12]. P. Michael E. Thomadakis, "The Architecture of the Nehalem Processor," The Architecture of the Nehalem Processor, p. 48, 2011.

[13]. S. Mittal and Nitin, "A New Approach to Directory Based Solution for Cache Coherence Problem," A New Approach to Directory Based Solution for Cache Coherence Problem, p. 5, 2015.

[14]. Tiwari, "Performance Comparison of Cache Coherence Protocol on Multi-Core Architecture," pp. 17-19, 2014.

[15]. Partha Kundu and Li-ShiuanPeh, "On-chip interconnects for multicores," Micro, IEEE, vol. 27, no. 5, pp. 3-5, Oct. 2007.

[16]. Michael R. Marty, Cache coherence techniques for multicore processors. Ph.D. Dissertation, University of Wisconsin - Madison, Madison, USA,2008.

[17]. Daniel J. Sorin, Mark D. Hill, and David A. Wood, A Primer on Memory Consistency and Cache Coherence. USA: Morgan and Claypool, 2011.

[18]. Saparon A, Razlan FN." Cache Coherence Protocols in Multi-Processor","InInternational conference on Computer Science and Information Systems (ICSIS), Dubai (UAE), pp. 17-18, Oct 2014.

[19]. Attada Sravanthi1, Ch. Rajasekhara Rao2, K. Krishnam Raju3, L. Rambabu4."Implementation of MESI protocol using VERILOG. International Research Journal of Engineering and Technology (IRJET). Volume: 06 Issue: 06, pp1763-1777.June 2019.

[20]. Handy, J. The Cache Memory Book. Academic Press, 1998.

[21]. Pavan Shree B. V, Mrs. Anitha V, "Design and implementation of direct mapped cache memory with same tag bit information", International journal of computer science and mobile computing, vol. 4, issue-6, pp. 978- 983. June 2015.

[22]. D.A. Patterison, J.I. Hennessy.Computer organization and design.4th edition.1998.

[23]. Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An evaluation of directory schemes for Cache coherence. In ACM SIGARCH Computer Architecture News, volume 16, pages 280–298. IEEE Compute Society Press, 1988.

[24]. Computer Architecture Lecture 31: Multiprocessor Correctness and Cache Coherence. Available at http://www.ece.cmu.edu.

[25]. P. T. a. L. C. Stewart., "Firefly: a Multiprocessor Workstation," In Proceedings of ASPLOS 11, vol. 37, pp. 164-172, 1987.

[26]. R. Fuchsen, "HOW TO ADDRESS CERTIFICATION FOR MULTICOREBASED IMA PLATFORMS: CURRENT STATUS AND POTENTIAL SOLUTIONS," Digital Avionics Systems Conference (DASC), 3 12 2010.

[27]. P. S. a. W. Zhang, "WCET Estimation of Multi-Core Processors with the MSI Cache," 2013

[28]. S. Alzahrani and H. Altwaijry, "Improved-MOESI Cache Coherence Protocol," Research Article - Computer Engineering and Computer Science, 2013.

[29]. Hadeel Sh. Mahmood, "VHDL Implementation of a Pipelined RISC Processor for Educational Purposes". MSc. Thesis, College of Electrical and Electronic Engineering Techniques, Baghdad, Iraq, 2014.

[30]. Computer Architecture Lecture 31: Multiprocessor Correctness and Cache Coherence. Available at http://www.ece.cmu.edu.

Mays K. Faeq. "MSI Protocol for Multicore Processors Based on FPGA." *International Journal of Engineering Science Invention (IJESI)*, Vol. 09(05), 2020, PP 44-55.