International Journal on Emerging Technologies **11**(3): 468-475(2020)

ISSN No. (Print): 0975-8364
ISSN No. (Online): 2249-3255

Design and Implementation of a United Multi-Core Memory Controller using AXI4-Lite Interface Protocol

***Ahmed Noami*[1], *B. Pradeep Kumar*[1] *and P. Chandrasekhar*[2]**
[1]*Ph.D. Scholar, Department of Electronics and Communication,*
College of Engineering, Osmania University, Hyderabad (Telangana), India.
[2]*Professor, Department of Electronics and Communication, College of Engineering,*
Osmania University, Hyderabad (Telangana), India.

(Corresponding author: Ahmed Noami)
(Received 26 February 2020, Revised 18 April 2020, Accepted 22 April 2020)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: Nowadays a multi-core SoC memory latency continues to become a critical bottleneck. Memory latency includes both on-chip memory and off-chip memory latency. The memory latency degrades the entire system performance of multi-core SoC while accessing the memory for write and read operations. Most previous studies treated the on-chip memory controller and the off-chip memory controller as independent stages. Without a clear vision for traffic between the two stages, as an example, while accessing SDRAM memory, unwanted scenario happen for precharge and activation row buffers in off-chip memory controller stage which increases the time and power. The main challenge design of any memory controller for multi-core processors is to decrease the latency while accessing the main memory for write and read operations which lead to improve the speed up of SoC design. In this work, a united multi-core SoC memory controller is proposed with burst mode capability using Advanced eXtensible Interface protocol (AXI4-Lite), to improve the entire system speed up of multi-core memory controller SoC. The proposed multi-core memory controller is designed by different Intellectual Property (IP) core and connects all these IP cores using the AXI4-Lite interface protocol to easily communicate and improve the system speed up. The memory controller design is implemented using System Verilog HDL, simulation and synthesis are done by using the Vivado tool and FPGA ZYNQ-7 ZC702 Evaluation Board (xc7z020clg484-1) accordingly with a maximum speed frequency of 100 MHz.

Keywords: On-chip memory, Off-chip memory, Memory controller, Advanced eXtensible interface.

Abbreviations: SoC, system on chip; FPGA, field-programmable gate array; HDL, hardware description language; Tcl, Tool Command Language; ACLK, clock signal; ARESET, reset signal; AWADDR, write address; ARADDR, read address; AWBURST, write address burst ; ARBURST, read burst address ; AWVALID, write address valid; ARVALID, read address valid; AWREADY, write address ready; ARREADY, read address ready; WDATA, write data; RDATA, read data; BVALID, write valid; BREADY, write ready; BRESP, write response; INCR, increment.

I. INTRODUCTION

A typical multi-core processors system has two types of random-access memory: on-chip memory, and off-chip memory. The on-chip memory usually consists of static random access memory (SRAM). On the other hand, off-chip memory usually consists of dynamic random access memory (DRAM). During write/ read access to/ from on-chip/ off-chip memories, the access latency of on-chip memory is less compared to that of off-chip memory.

The main reasons for high access latency in the off-chip memory are to store the data in the capacitors as charges and to refresh the data every a few cycles [1]. Each core machine consists of one processor, two or more levels of on-chip memory, off-chip memory and Input/ Output (I/O) devices. Levels of on-chip memory relate to the size and distance from the processor which displays the memory hierarchy, for example accessing data from the first level on-chip memory faster than accessing it from the second level, and so on. Consequently, the use of on-chip memory reduces the Memory Access Time (MAT) and resulting in a better performance [2].

From the miss requests available in memory controller' buffer, the memory controller based on scheduling policy selects only one request to access the memory in every clock cycle. Selected request to access the memory is sent to the command generators stage. This stage translates the memory request to commands to be able to access the off-chip memory for write/ read in the proper way. The data storage in off-chip memory is organized as multiple memory hierarchies which are represented by ranks, banks, rows, and columns respectively. The memory controller can manage the parallel memory accesses at ranks and banks memory hierarchies and only one-row buffer can be active in each bank. To write/read a column, it must be the target row of this column open in the row buffer before performing any actual write/ read access (row activate operation). The bank must be closed target row after write/ read completed (precharge operation) [3].

The traditional memory subsystem architecture of multi-core processors is shown in Fig. 1. The L3-level on-chip memory stage is the first access point of multi-core memory access traffics. The miss requests of L3-level on-chip memory are forwarded to the memory controller where these miss requests are buffered in the memory controller's transaction queue and waiting to be

scheduled to the off-chip memory for write/ read operations.
The two controllers of L3-level on-chip memory and off-chip memory as shown in Fig. 1 have seemed as though separate units that make incorrect final scheduling ruling. The forwarded L3-level on-chip memory miss requests are invisible at the off-chip memory controller side, as well as the amplified states of the row-buffer are invisible at L3-level on-chip memory controller side. These limited miss requests visibility on both sides oftentimes heads towards incorrect scheduling decisions. Increase L3-level on-chip memory miss requests rate by multi-core processors traffic makes problem by bringing more traffic miss requests to the off-chip memory controller. Therefore, the two stages on-chip memory controllers lead to increase latency while multi-core processors accessing the memory for write and read operations. However, in this paper, we designed a united multi-core memory controller to decrease system latency that leads to improve the system speed up.

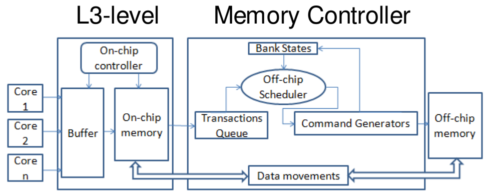


Fig. 1. Traditional memory subsystem of multi-core processors.

II. MOTIVATION AND LITERATURE

Architectures controllers of the L3-level on-chip memory and the off-chip memory have been discussed widely in recent years. All previous works were focused on the DRAM-aware management of L3-level on-chip write backs. However, L3-level on-chip memory misses at run time can also impact the performance of the scheduling process as shown in Fig. 2. Assume that A and B are two miss memory requests and both are waiting at the L3-level on-chip memory side. In the same off-chip memory bank (*K*) and different row buffers (*R*1 & *R*2 respectively) the target addresses of the two miss memory requests A and B are located. Initially as shown in Fig. 2 the row *R*2 of memory bank *K* is active. If both A and B requests existing at the on-chip memory side and request A is first served by the L3-level on-chip memory controller, it will arrive earlier than request B at the transaction queue. Without knowing at the same time by the existence of request B in the L3-level on-chip memory stage, the memory scheduler precharges row *R*2 and activate the row buffer of miss request A. During miss request B arrives next, the scheduler will precharge row *R*1 and then re-activate *R*2.
Most previous works tackle these two independent memory controller stages fabrics that lead to unwanted scenario happen for precharge, activation row buffers at the off-chip memory controller stage which increase the latency for multi-core processors while accessing the main memory for write and read operations [4-8]. Xilinx proposed a united memory controller using the AXI4-Lite interface protocol that handles both independent memory controller stages [10]. However, it can handle only a single-core processor. The design was proposed for one core processor with two write and two read operations to/ from the memory, which means that the single-core processor write two different 32-bits data to two different memory locations and after a nanoseconds of time the single-core processor read the same data. This scenario of write and read operations leads to more time to complete the two write and two read operations. The main contribution of this work is to write and read operations for single-core processor in a parallel way to improve the speed up of existing work and then design a united multi-core memory controller using the AXI4-Lite interface protocol that improves the speed up (decrease latency) while multi-core processors accessing the memory for write and read operations. All the cores can write/ read to/ from the main memory at the same time in parallel to improve the speed up of the entire SoC design.

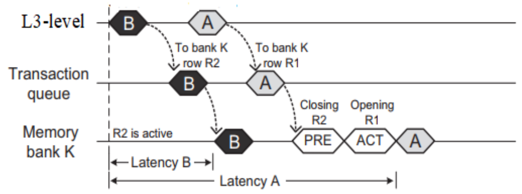


Fig. 2. Precharge and Activation Operations [9].

III. PROPOSED MODEL

Fig. 3 illustrates the proposed design model of the multi-core memory controller SoC. Different Intellectual Property (IP) exists in the design which represents all components of our multi-core memory controller model. The first IP is AXI verification (AXI VIP) [11]. It is an IP core using to initiate a write and read transactions as single or multi-core processors with different interface protocol modes such as AXI3, AXI4-Lite, and AXI4. In this paper, we used this IP core to initiate a write and read transactions for single and multi-core processors with AXI4-Lite interface protocol. The second IP is AXI Block RAM (BRAM) Controller [12]. It is a united memory controller that receives requests and manages them for access to the off-chip memory. This memory controller can also support different interface protocols such as AXI3, AXI4-Lite, and AXI4. We used four AXI BRAM Controllers with AXI4 interface to manage requests from four-core processors. Each core processor can manage by an independent AXI BRAM Controller. The third IP is Block Memory Generator [13]. It is an IP core that creates the BRAM which represents a portion of the off-chip memory which only one processor can access it. Regardless of the type of the off-chip memory, we used four BRAMs which represent four portions of the off-chip memory and each processor can access its address space (BRAM). The fourth IP is AXI Interconnect [14]. The AXI Interconnect IP core allows connects one AXI master or more and one AXI slave or more, which can be different kinds of interface protocol, clock domain, and data width. We used this IP to connect one master (AXI VIP) and multiple slaves (AXI Memory Controller). Inside the AXI Interconnect IP, it is available also data buffer in different sizes which are working as L3-level on-chip memory between one core processor or multi-core processors and memory controller. These data buffers accommodate the data movement between one core processor or multi-core processors and off-chip memory (BRAM). The last IP is Processor System Reset [15]. We used this IP to reset the single-core processor or multi-core processors and other different IPs available in our design.

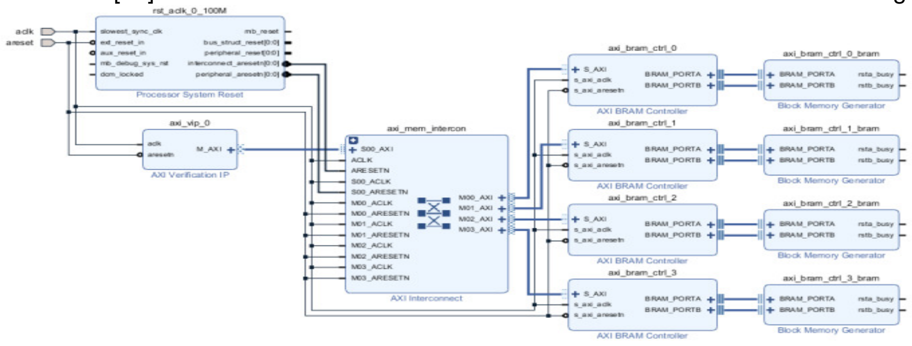


Fig. 3. Proposed Model.

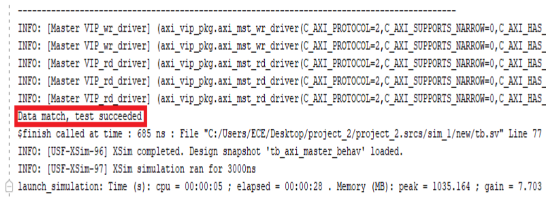
In this work, the proposed design is implemented in single and multi-core processors for different modes of write and read operations as shown in the simulation results section IV.

A. Write Operation

The single and multiple write operations for single and multi-core processors using the AXI4-Lite interface protocol are shown in the flowchart in Fig. 5. The AXI4-Lite interface has three independent channels for write operation: write address channel, write data channel, and write response channel [16]. The flowchart shows the all three independent channel signals that executes the write operation of single-core or multi-core processors in the proper way. At the beginning, if the clock and reset signals are high, the single-core or multi-core processors (Master) can start the write operation when the signals of write address channel AWVALID and AWREADY are high, which represents that the write address from the master is valid and the memory controller (Slave) ready to receive the write address from the master. The signal AWBURST from master to salve indicates that the write operation will be in the burst mode and represented by binary value 2'b01. The variable W indicates the memory address register. If the register is equal to zero this means that the write operation will be for the start address, otherwise, add the digit four (4) to the content of the register W which indicates the next address. The start addresses according to our design are C0000000, C2000000, C4000000, and C6000000 of the four core processors respectively. The write data operation is transferred when the write channel signals WVALID and WREADY are high, which indicates that the data transmitted from master to slave is valid and the salve is prepared to receive the data transmitted from the master. WDATA represents the data transferred from the master to the slave. The last signal BVALID, BREADy, and BRESP represents the write response channel of the write operation. BVALID signal sends from slave to master that indicates all data are received, BREADY signals send from master to slave that indicates the master is ready to receive a response about the data sent and BRESP signals just indicates the status of the transaction.

B. Read Operation

The single and multiple read operations for single and multi-core processors using the AXI4-Lite interface protocol are shown in the flowchart in Fig. 6. The flowchart shown that AXI4-Lite interface protocol has only two independent channels for read operation: read address data channel [16]. The ARVALID, ARREADY, ARBURST signals represent the read address channel, and RVALID and READY signals represent the read data channel. The read transactions of the AXI4-Lite interface protocol signal details are the same write transactions mentioned in the previous section.



```
---------------------------------------------------------------------------------------
INFO: [Master VIP_wr_driver] (axi_vip_pkg.axi_mst_wr_driver(C_AXI_PROTOCOL=2,C_AXI_SUPPORTS_NARROW=0,C_AXI_HAS_
INFO: [Master VIP_wr_driver] (axi_vip_pkg.axi_mst_wr_driver(C_AXI_PROTOCOL=2,C_AXI_SUPPORTS_NARROW=0,C_AXI_HAS_
INFO: [Master VIP_rd_driver] (axi_vip_pkg.axi_mst_rd_driver(C_AXI_PROTOCOL=2,C_AXI_SUPPORTS_NARROW=0,C_AXI_HAS_
INFO: [Master VIP_rd_driver] (axi_vip_pkg.axi_mst_rd_driver(C_AXI_PROTOCOL=2,C_AXI_SUPPORTS_NARROW=0,C_AXI_HAS_
INFO: [Master VIP_rd_driver] (axi_vip_pkg.axi_mst_rd_driver(C_AXI_PROTOCOL=2,C_AXI_SUPPORTS_NARROW=0,C_AXI_HAS_
Data match, test succeeded
$finish called at time : 685 ns : File "C:/Users/ECE/Desktop/project_2/project_2.srcs/sim_1/new/tb.sv" Line 77
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_axi_master_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 3000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:28 . Memory (MB): peak = 1035.164 ; gain = 7.703
```

Fig. 4. Vivado Tcl Console Command Massage.

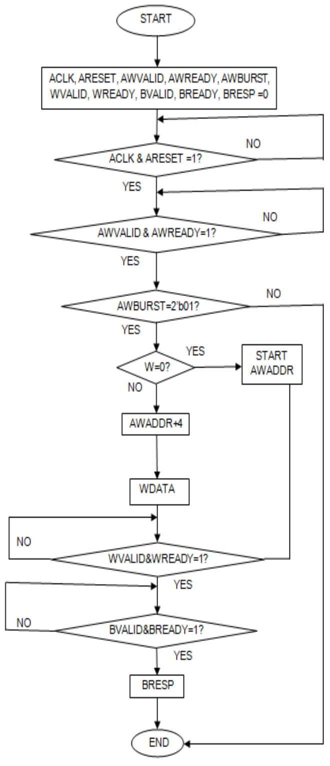


Fig. 5. Flowchart of Single and Multi Write Operations.

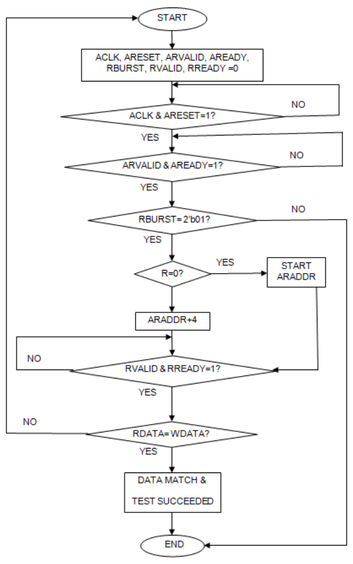


Fig. 6. Flowchart of Single and Multi Read Operations.

At the end of read operation as shown in the flowchart in Fig. 6, the comparison step between the write and read transactions. If data read is same data write, then the message "data matched and test succeeded" printed in the Tcl Console Command. A sample message of the simulation results of our design is shown in snapshot in Fig. 4 below. Otherwise, try to read operation again to match the data write.

IV. SIMULATION RESULTS

Fig. 7 (a) and (b) illustrate the snapshot of the simulation results of a united memory controller for single-core processor in various operation modes. Fig. 7 (a) shows the two write and two read operations for single-core processor [10]. The two write and two read operations are completed at time 585ns. The transactions are done in the normal way which the single-core processor write two different 32–bits data (abcde000 and abcde001) into two different 32-bits memory location (C0000000 and C0000004) then after a nanoseconds of time the single-core processor read the data. Our proposed model executes the two write and two read operations in a parallel way which leads to reduce the transaction time. The two write and two read operations of our proposed model for single-core processor completed at time 565ns as shown in Fig. 7 (b). In the Fig. 8 (a) and (b) show the simulation results of a united memory controller for single-core processor with three write and three read operations. In the figure 8a, the transactions are done in the normal way which the single-core processor write three different 32–bits data (abcde000, abcde001, and abcde002) into three different 32-bits memory locations (C0000000, C0000004, and C0000008) then after a nanoseconds of time the single-core processor read the data.

The three write and three read operations completed at time 725ns. However, the three write and three read operations of our proposed model for single-core processor completed in a parallel way at time 685ns as shown in Fig. 8 (b). In the other simulation results, we increased the number of core processors and number of write and read operations (united memory controller for multi-core processors) using the same two different ways mentioned above.

In the Fig. 9 (a) and (b) show the simulation results of a united memory controller for two-core processors with two write and two read operations for each core processor using the same two different ways mentioned above. In the normal way, the first core processor write two different 32–bits data (abcde000 and abcde001) into two different 32-bits memory locations (C0000000 and C0000004) and then after a nanoseconds of time the first core processor read the data.

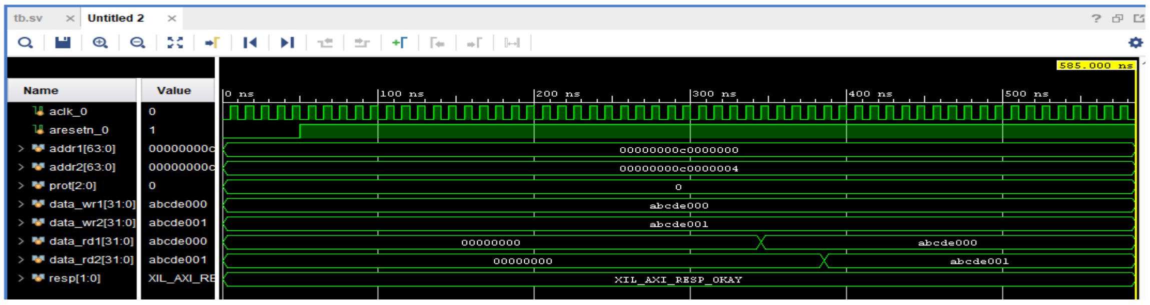


Fig. 7 (a) 2 write and 2 read transactions of single core processor [10].

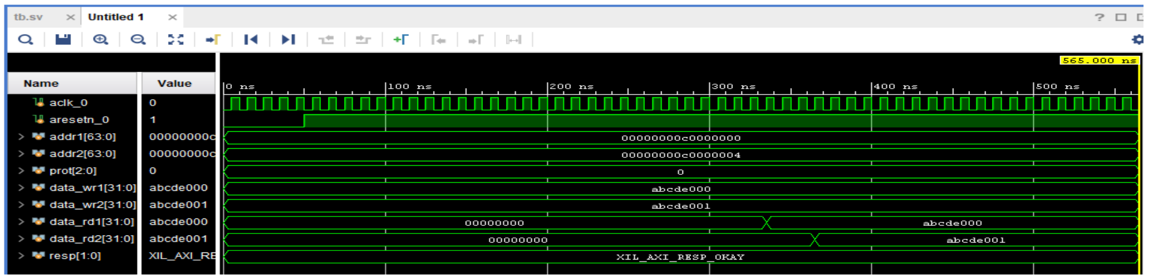


Fig. 7 (b) 2 write and 2 read transactions of single core processor in parallel.

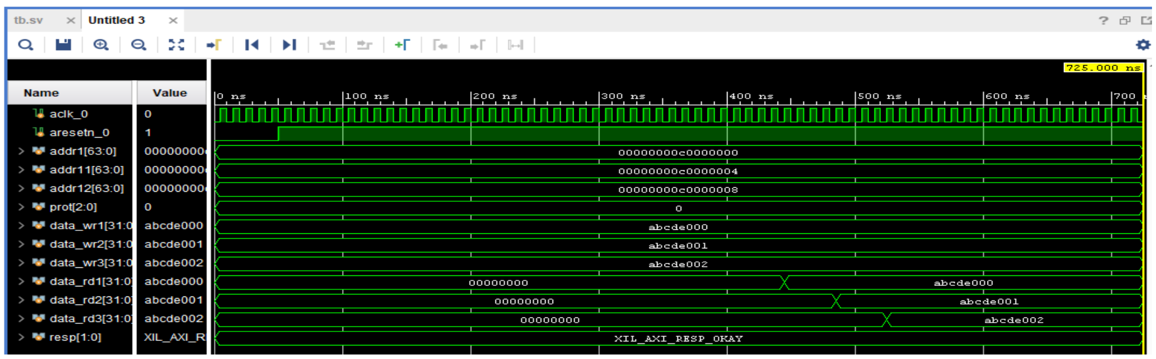


Fig. 8 (a) 3 write and 3 read transactions of single-core processor.

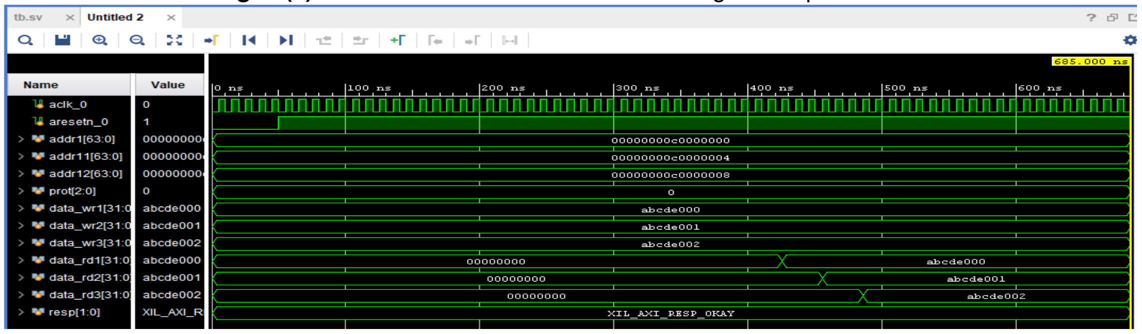


Fig. 8 (b) 3 write and 3 read transactions of single-core processor in parallel.

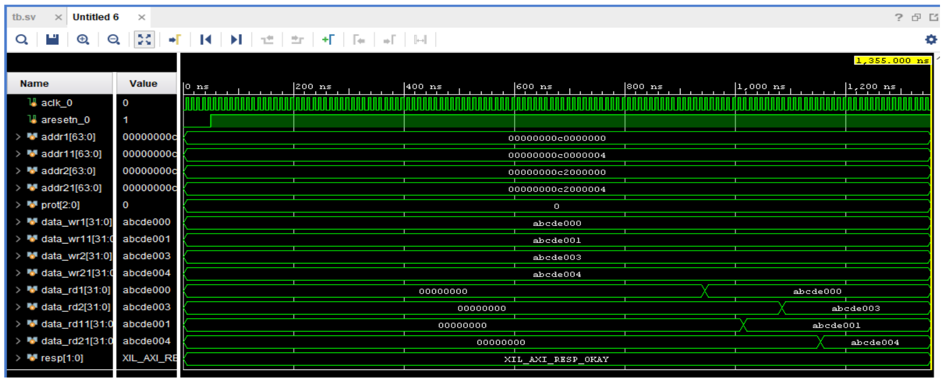


Fig. 9 (a) 2 write and 2 read transactions of two-core processors.

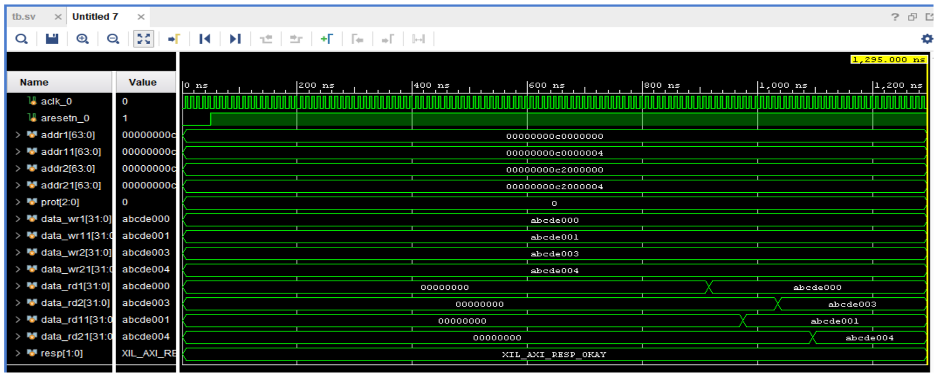


Fig. 9 (b) 2 write and 2 read transactions of two-core processors in parallel.

The second core processor write two different 32–bits data (abcde003 and abcde004) into two different 32-bits memory locations (C2000000 and C2000004) then after a nanoseconds of time the second core processor read the data. This two write and two read operations completed at time 1355ns. In a parallel way (our proposed), the two-core processors can write their two different 32-bits data (abcde000 and abcde001) for the first core processor and (abcde003 and abcde004) for the second core processor into two different 32-bits memory locations (C0000000 and C0000004) for the first core processor and (C2000000 and C2000004) for the second core processor at the same time respectively. This two write and two read operations completed at time 1295ns. In the Fig.10a and figure 10b show the simulation results of two-core processors with three writes and three read operations for each core processor using the two different ways.

In the normal way, the first core processor write three different 32–bits data (abcde000, abcde001, and abcde002) into three different 32-bits memory locations (C0000000, C0000004, and C0000008) and then after a nanoseconds of time the first core processor read the data. The second core processor write three different 32–bits data (abcde003, abcde004 and abcde005) into three different 32-bits memory locations (C2000000, C2000004, and C2000008) then after a nanoseconds of time the second core processor read the data. This three write and three read operations completed at time 1695ns. In a parallel way (our proposed), the two-core processors can write their three different 32-bits data (abcde000, abcde001, and abcde002) for the first core processor and (abcde003, abcde004, and abcde005) for the second core processor into three different 32-bits memory locations (C0000000, C0000004, and C0000008) for the first core processor and (C2000000, C2000004, and C2000008) for the second core processor at the same time respectively. This three write and three read operations completed at time 1595ns. The other remaining simulation results of a united memory controller for one/ two/ three/ four-core processors with two/ three/ four write and two/ three/ four read operations using the two different ways are directly written into Table 1. It is reported from Table 1 that our proposed model improves the speed up of write and read operations. In a united memory controller for single-core processor with two/ three/ four write and read operations, our model improves the speed up of write and read operations by 20ns, 40ns, and 60ns respectively. In two-core processors with two/ three/ four write and read operations for each core processor, our model improve the speed up of write and read operations by 60ns, 100ns, and 140 ns respectively.

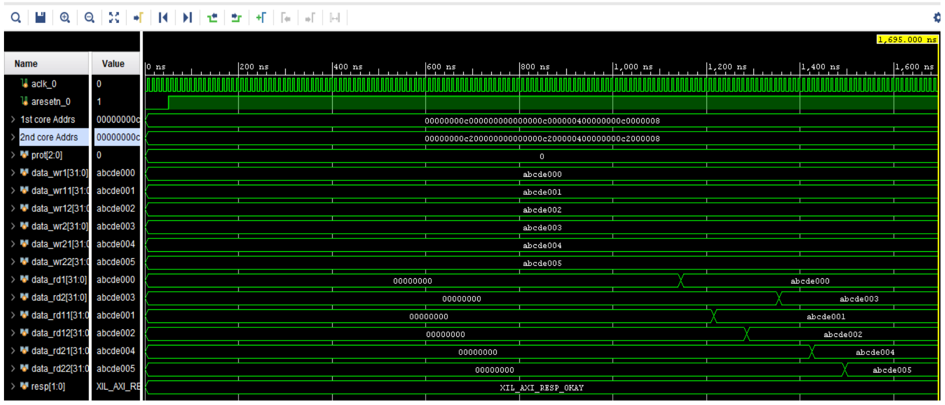


Fig. 10 (a) 3 write and 3 read transactions of two-core processors.

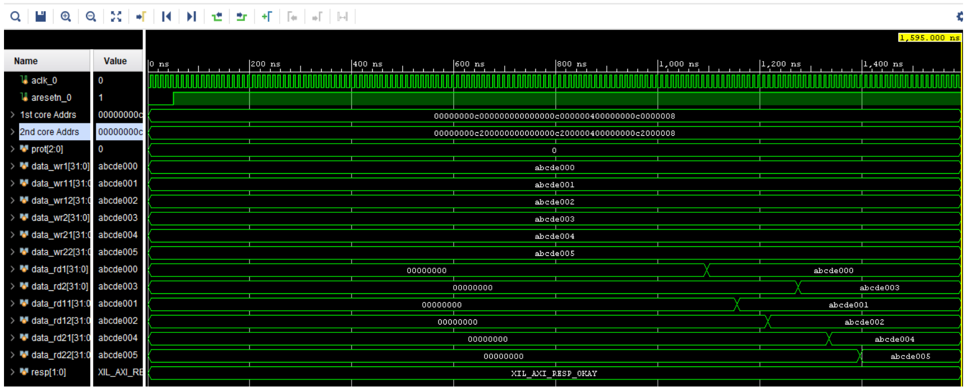


Fig. 10 (b) 3 write and 3 read transactions of two-core processors in parallel.

Table 1: Comparison between Number of Core Processors and Number of Transactions.

Core Processor / Transactions	Single-Core Processor	Two-core Processors	Three-core Processors	Four-core Processors
2 Write & 2 Read [10] (Normal Way)	585 ns	1355 ns	1695 ns	2055 ns
2 Write & 2 Read (Our Model)	565 ns	1295 ns	1595 ns	1895 ns
3 Write & 3 Read (Normal Way)	725 ns	1695 ns	2205 ns	2735 ns
3 Write & 3 Read (Our Model)	685 ns	1595 ns	2045 ns	2495 ns
4 Write & 4 Read (Normal Way)	865 ns	2035 ns	2715 ns	3395 ns
4 Write & 4 Read (Our Model)	805 ns	1895 ns	2495 ns	3095 ns

In three-core processors with two/ three/ four write and read operations for each core processor, our model improves the speed up of write and read operations by 100ns, 160ns, and 220ns respectively. In four-core processors with two/ three/ four write and read operations for each core processor, our model improves the speed up of write and read operations by 160ns, 240ns, and 300ns respectively.
We observed from the simulation results that the normal way [10], which execution the write and read operations leads to increase the latency while the multi-core processors SoC and write and read operations are increases. This normal way it seems undesirable for many multi-core processors SoC applications that need the speed up to improve the entire performance of the design.
Also, we observed from the simulation results that the parallel way, our model, which execution the write and read operations decrease the latency for write and read operations of multi-core processors which lead to improve the speed up of SoC design. This way desirable for many multi-core processors SoC applications which need to execute so many write and read operations at the same time and improve the entire performance of SoC design.

V. REAL-TIME DESIGN ANALYSIS

Debugging multi-core memory controller is done on FPGA ZYNQ-7 ZC702 Evaluation Board (xc7z020clg484-1). At the beginning of the debugging design on FPGA, we replaced the VIP IP core that initiated all write and read AXI4-Lite transactions of single and multi-core processors in the simulation stage by JTAG-to-AXI IP core [17]. Because VIP IP core is supporting only simulation stage of the design and VIP IP core is replaced by wires after synthesis design. The JTAG-to-AXI IP core initiates the real-time write and read AXI4-Lite transactions at debugging design stage on FPGA by using Tcl console command of the Vivado tool. The Tcl console command of write AXI4-Lite transaction written in the form such "create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -address xxxxxxxx -data {zzzzzzzz} -type write". This command indicates the type of transaction, address then data. For the read AXI4-Lite transaction, the command written such "create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] –address xxxxxxxx -type read". This command indicates the type of transaction and the address only. These Tcl console commands for both write and read transactions are already supports the INCR type of the burst mode for only one burst length data transfer with 32 bits width. Finally, we used also different Tcl console commands to run the write and read operations on FPGA hardware. These commands are run_hw_axi [get_hw_axi_txns write_txn] for write operations and run_hw_axi [get_hw_axi_txns read_txn] for read operation.

Table 2: FPGA Utilization Summary.

Logic Utilization	Available	Used	Utilization Percentage
Slice LUTs	53200	4658	8.75%
Slice Registers	106400	2842	2.67%
Slice	13300	1932	14.5%
LUT as Memory	17400	2851	16.38%
Block RAM Tile	140	14	10%
Bounded IOB	200	2	1%

In the existing model there is no mentioned for the FPGA device utilization. However all the logics hardware utilization summary such as lookup tables, registers, slice, lookup tables as memory, block RAM and inputs/ outputs of the ZYNQ-7 ZC702 Evaluation Board (xc7z020clg484-1) of our proposed model are shown in Table 2.

VI. CONCULSION

In this paper a united multi-core memory controller using the AXI4-Lite interface protocol is proposed to improve the SoC speed up. The proposed model is simulated for one/ two/ three/ four-core processers with two/ three/ four write and read operations for each core processor. Our design improved the speed up for one/ two/ three/ four-core processors with two/ three/ four write and read operations. It is shown from simulation results that our design decreased the access time latency of the write and read operations of single and multi-core processors. For one/ two/ three/ four-core processors with two write and read operations, the latency is decreased by 3.42%, 4.43%, 5.9%, and 7.78% respectively. For two-core processors with two/ three/ four write and read operations the latency is decreased by 4.43%, 5.9%, and 6.94% respectively, etc. The design is implemented using System Verilog HDL. The simulation and synthesis are done by using Vivado tool and FPGA ZYNQ-7 ZC702 Evaluation Board (xc7z020clg484-1) accordingly.

VII. FUTURE SCOPE

AXI4-Lite interface protocol has a limitation features for data write and read operations. It is supported only by fixed 32-bits data transaction size and one data burst mode for each transaction.

Several SoC design needs interface supports variable data size and different burst mode such as AXI4 full memory-mapped interface protocol.

ACKNOWLEDGEMENT

This work has been supported by the Indian Council for Cultural Relations (ICCR), India.

Conflict of Interest. No.

REFERENCES

[1]. Hussain, T., (2014). *A Novel Access Pattern-based Multi-core Memory Architecture* (Doctoral dissertation, Departament d'Arquitectura de Computadors, universitat politècnica de catalunya). Retrieved from https://upcommons.upc.edu/handle/2117/95566.
[2]. Sirhan, N., Serhan, S., (2018). Multi-Core processors: Concept and Implementations. *International Journal of Computer Science & Information Technology, 10*(1): 1-10.
[3]. Tigadi, A., & Guhilot, H. (2018). Design and Implementation of a DDR2 SDRAM Controller for Audio Data on a Reconfigurable Platform. *International Journal of Engineering and Manufacturing, 8*(5), 32-48.
[4]. Rixner, S., Dally, W., Kapasi, U., Mattson, P., & Owens, J. (2000). Memory access scheduling. The 27th *International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, 128-138.
[5]. Kaseridis, D., Stuecheli, J., & John, L. (2011). Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era. The 44th Annual

IEEE/ACM*I International Symposium on Microarchitecture*, 24-35.

in heterogeneous systems. *The 39th Annual IEEE International Symposium on Computer Architecture,* 416-427.

[7]. Lee, J., & TAP, H. (2012). A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture. *IEEE International Symposium on High-Performance Comp Architecture,* 1-12.

[8]. Mekkat, V., Holey, A., Yew, P., & Zhai, A. (2013). Managing shared last-level cache in a heterogeneous multicore processor. *The 22nd International Conference on Parallel Architectures and Compilation Techniques*, 225-234.

[9]. Song, Y., Alavoine, O., & Lin, B. (2018). Row-Buffer Hit Harvesting in Orchestrated Last-Level Cache and DRAM Scheduling for Heterogeneous Multicore Systems. *Design,* Automation & Test in Europe Conference & Exhibition (DATE), 779 – 784.

[10]. Confluence, Using the AXI4 VIP as a master to read and write to a AXI4-Lite slave interface, (2020). [online]. Available at xilinx-wiki.atlassian.net.

[11]. AMBA AXI Verification IP, LogiCORE IP Product Guide PG267 (v1.1) (2019). [Online]. Available at http://www.xilinx.com.

[6]. Ausavarungnirun, R., Chang, K., Subramanian, L., Loh, G., & Mutlu, O., (2012). Staged memory scheduling: Achieving high performance and scalability

[12]. AMBA AXI BRAM Controller, LogiCORE IP Product Guide PG078 (v4.1) (2019). [Online]. Available at http://www.xilinx.com.

[13]. AMBA AXI Block Memory Generator reference guide PG058 (v8.4) (2019). [Online]. Available at http://www.xilinx.com.

[14]. AMBA AXI Interconnect, LogiCORE IP Product Guide PG059 (v2.1) (2019). [Online]. Available at http://www.xilinx.com.

[15]. AMBA AXI Processor System Reset Module, LogiCORE IP Product Guide PG164 (v5.0) (2015). [Online]. Available at http://www.xilinx.com.

[16]. Sainath Chaithanya, A., Sulthana, S., Yamuna, B. & Haritha, Ch (2020). Design of AMBA AXI4-Lite for Effective Read/Write Transactions with a Customized Memory. *International Journal on Emerging Technologies*, *11*(1), 396–402.

[17]. AMBA JTAG to AXI Master, LogiCORE IP Product Guide PG174 (v1.2) (2016). [Online]. Available at http://www.xilinx.com.

How to cite this article: Noami, A., Kumar, B. P. and Chandrasekhar, P. (2020). Design and Implementation of a United Multi-Core Memory Controller using AXI4-Lite Interface Protocol. *International Journal on Emerging Technologies*, *11*(3): 468–475.