UVM based Verification Environment for Performance Evaluation of DDR4 SDRAM using Memory Controller

Nausha Kotia[1] Mr. Prashant D. Karandikar[2] Mr. Gardas NareshKumar[3]
[1]Department of Electronics Engineering
[1]GTU PG SCHOOL Ahmedabad, India
[2]Aurora Copper Systems LLP Pune, India [3]CDAC-ACTS, Pune

Abstract— In digital Design, Verification plays a vital role as most of the time is engaged in verification. So ASIC or SOC verification companies adopted several methodologies for re-usability purpose. The latest methodology adopted is UVM due to its advantages. This paper exhibits Performance checking of DDR4 memory and controller using UVM based on work load, i.e. achieved at RTL level with the use of Scheduling algorithm. To verify the functionality and timing accuracy of memory controller IP, test-bench is constructed at TLM level.

Key words: UVM, Verification Environment, TLM, AXI4, DDR4.

I. Introduction

Performance parameters plays a vital role in memory controller for the applications like Automobile, Mobile, vehicular applications like infotainment etc. which is evaluated here using UVM (addresses re-usable and robust verification environment), depending on the workload of memory controller. The design is having inconsistency between functionality and time to market. So it will be overcome with verification environment provided by latest verification methodology followed by Accellera in 2011.   Read and write response latency and bandwidth are parameters that require at most consideration.

Cross performance estimation is obtained with respect to the models, with the help of design level verification environment based on UVM.

This paper describes test-bench architecture which encapsulates several verification components with particular functionalities and their interconnections among them. Section II carries out detail depiction of UVM based verification environment. Section III depicts DDR4 SDRAM controller architecture, Section IV explains DDR4 advantages, Section V shows Results of Simulation of AXI4 read and write transactions and measurement of latency based on test-cases practiced. Conclusion is stated in Section VI.

II. Universal Verification Methodology

A. Introduction to UVM:

To construct an adequate and designed verification environment [11] the methodology used is UVM. It makes an ease by providing base class library of system Verilog language (used for design and verification). To provide re-usability, it splits the design which builds verification environment and the stimulus applied to the sequences as shown in Figure1.

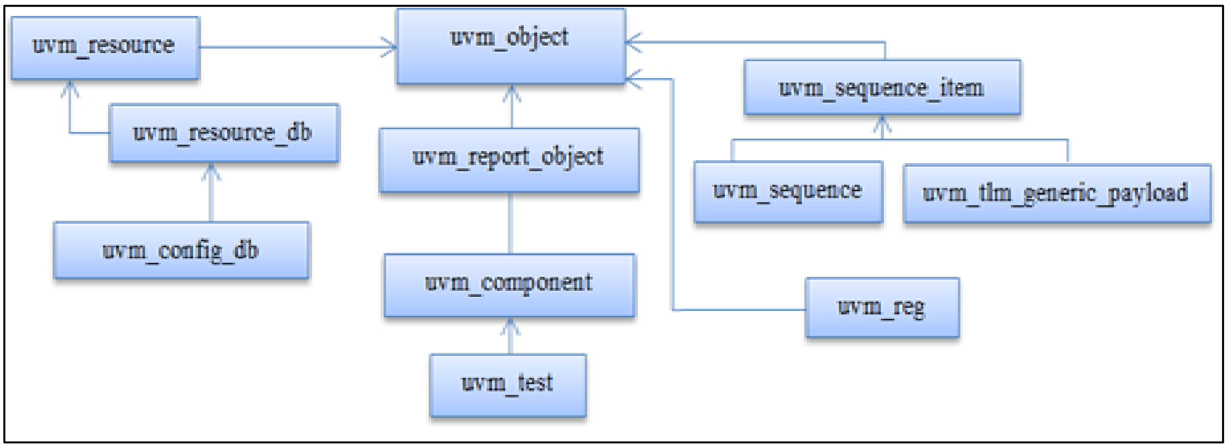


Fig. 1: Class Hierarchy of UVM [11]

UVM class library hierarchy gives several advantages:

1) Inbuilt aspects increases to meet the requirements of test phases for verification, printing application, to provide factory methods, the transaction level modeling (TLM) ports required for interconnection among components and utility macros too [7].

2) The correct implementation of code based on UVM approach leads to an efficient results using base class, as it generates interrelated components behavior using parent class with increase in readability. Fig 1, illustrates that parent class is an uvm_object for every other uvm_classes. To build the test-bench architecture, components required is extended from the uvm_components. And many other classes provide its individual functionalities inside the hierarchy.

Fig 2 depicts the verification environment using UVM, resides the various components:

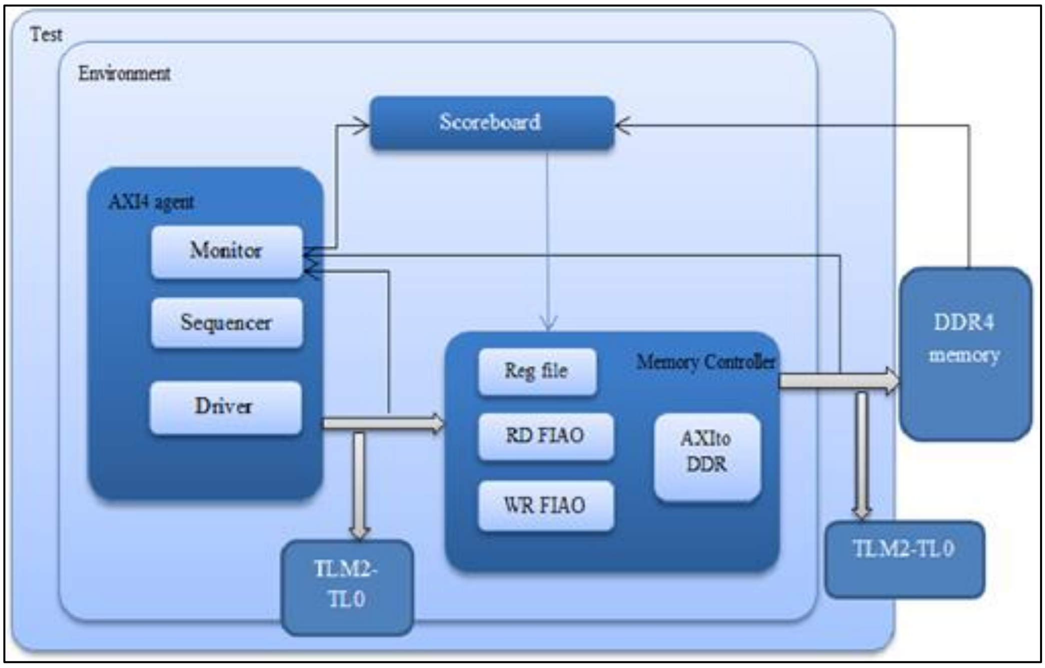


Fig. 2: UVM based Verification Environment

TLM builds up the communication interface i.e. TLM2-TL0 adapter for interconnection between components which is achieved at the transaction level, to gain the advantages like less time to market, effort required is less, and most importantly is re-usability for construction of adequate environment. The test, Env and DUT are the essential blocks encapsulated inside the top module required to build the test-bench.

B. Test:

Test is the top most verification component which encapsulates all other blocks. UVM test performs three important functions i.e. Instantiation of environment, composition of Environment (by applying config_db or the factory methods), and using environment UVM sequences is called, with that stimulus is practiced to the DUT. Using the uvm_test base class library, Test class is being extended.

C. Environment:

From the uvm_env base class library environment class is extended. From the Fig2. Environment consists of several blocks like objects of AXI4 agent, Monitor, Scoreboard, Adapter for interconnection components vary depending on design all these to targeting on DUT and keeping handle of interfacing.

D. Register File:

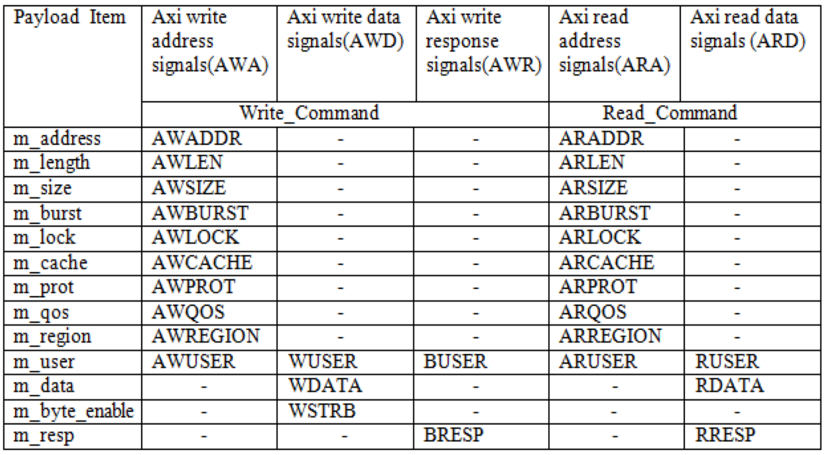
The register file type is accomplished with the help of register model, and in the scope of declaration, name conceived is exclusive. Using the class of register file an appropriate generation of the uvm_object_utils () macro is achieved and instantiation of register files are accomplished within the build() method. With the uvm_reg base class elongation it is constructed.

E. Agent:

The Agent includes the objects of driver, sequencer and monitor only to deal with a DUT interface. Using TLM port and export connection, a stimulus flow is exercised to DUT from the driver which is urged by sequencer by employing connect phase. Agent class is elongated from uvm_agent base class and include interface handle to carry to the other components.

F. Sequencer:

A sequencer restrians the items for execution which is applied to the driver and restore a random data item from driver consequent to the request. Sequencer class extends from uvm_sequence_item. The AXI phases- begin_req, end_req, begin_data, end_data, begin_resp, end_resp leads to carrying the transactions developed from sequence to driver.



Payload Item	Axi write address signals(AWA)	Axi write data signals(AWD)	Axi write response signals(AWR)	Axi read address signals(ARA)	Axi read data signals (ARD)
	Write_Command			Read_Command	
m_address	AWADDR	-	-	ARADDR	-
m_length	AWLEN	-	-	ARLEN	-
m_size	AWSIZE	-	-	ARSIZE	-
m_burst	AWBURST	-	-	ARBURST	-
m_lock	AWLOCK	-	-	ARLOCK	-
m_cache	AWCACHE	-	-	ARCACHE	-
m_prot	AWPROT	-	-	ARPROT	-
m_qos	AWQOS	-	-	ARQOS	-
m_region	AWREGION	-	-	ARREGION	-
m_user	AWUSER	WUSER	BUSER	ARUSER	RUSER
m_data	-	WDATA	-	-	RDATA
m_byte_enable	-	WSTRB	-	-	-
m_resp	-	-	BRESP	-	RRESP

Table 1: Axi4 Payload [12]

G. Driver:

A driver class is elongated from uvm_component base class. Forward call is made at driver, using interface handle and AXI signals is passed from sequence to driver using TLM2-TL0 adapter depending on the phases. Inside the build phase using get(), handle of interface is called. So handshake process is carried out between sequence and driver within the run phase of driver.

H. Monitor:

Monitor class is elongated from uvm_component base class. The sampling of DUT signals is carried out inside the monitor but it won't drive any signals. Based on the coverage information and transactions collected monitoring takes place inside the monitor.

I. Scoreboard:

Scoreboard class is elongated from uvm_component base class. Through agent analysis port it checks the nature of DUT, and carried out expected results from the transactions received and then comparing it with the actual output.

III. DDR4 CONTROLLER

DDR4 is the first JEDEC DRAM which is appropriate with 3-D standard unlike other DRAMs [3], with lots of improvements i.e. huge capacity, greater speed, efficiency were all are very crucial in computing platforms, embedded application and many more. DDR4 with higher data transfer rates provides much better performances on time. It also strengthens the reliability and power efficiency.

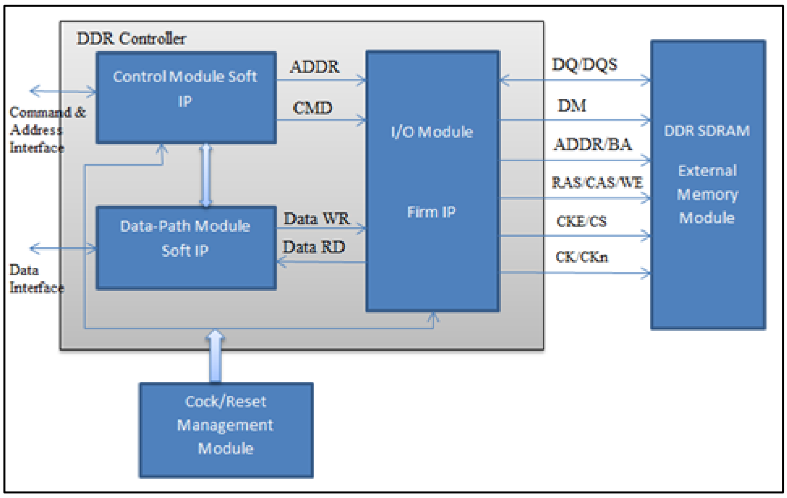


Fig. 3: Architecture of DDR4 Controller

DDR4 controller architecture contains distinct blocks interconnecting with one other as shown in Fig.3.

DDR4 controller is the DUT for performance checking, associating AXI4 agent and DDR4 memory and it encapsulates reg file, RDFIAO, WRFIAO, Queue, axi2ddr translator and PLL for clock generation, DUT is having clock and reset to function. The input to RDFIAO and WRFIAO is AXI read and write signals which are passes through queue from AXI4 interface and it is translated to DDR4 using axi2ddr (Arbiter and scheduler) as shown in Fig 4.

The write latency is measured, during AXI4 write transactions, were write request arrives at axi2ddr once it get translated page is open at DDR4 memory, it waits for AL+CWL cycle time and results in data in terms of response. The difference within the request and response arrived is measured.

Equivalent to write latency, for AL+CL cycle time measurement of read latency is achieved.
For adjustment in matching the frequency of an input signals PLL clock is used.

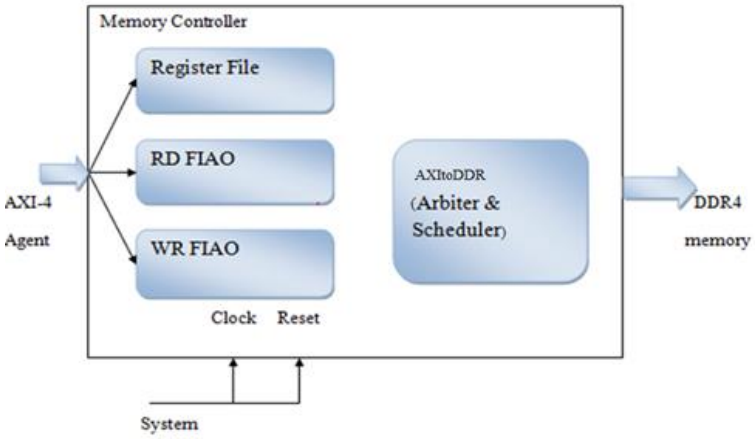


Fig. 4: Design Under Test

IV. DDR4 ADVANTAGES

- DDR4 is having larger size memory subsystem with greater improvement.
- Due to parallel Bank group there is an ease page-open individually, which provides higher optimization.
- More banks and Faster burst accesses so there is a reduction in power demand.
- It is having write levelling and initialization support.
- Shifting to higher bandwidth than the other DRAMs.

V. SIMULATION RESULTS

After applying test-cases, using Model-Sim simulator, AXI4 transactions are observed for every channel According to AXI4 protocol specifications communication is carried out using VALID READY handshake pair [12], as shown in below figures.

A. *AXI4 Write and Read Channel Signals:*

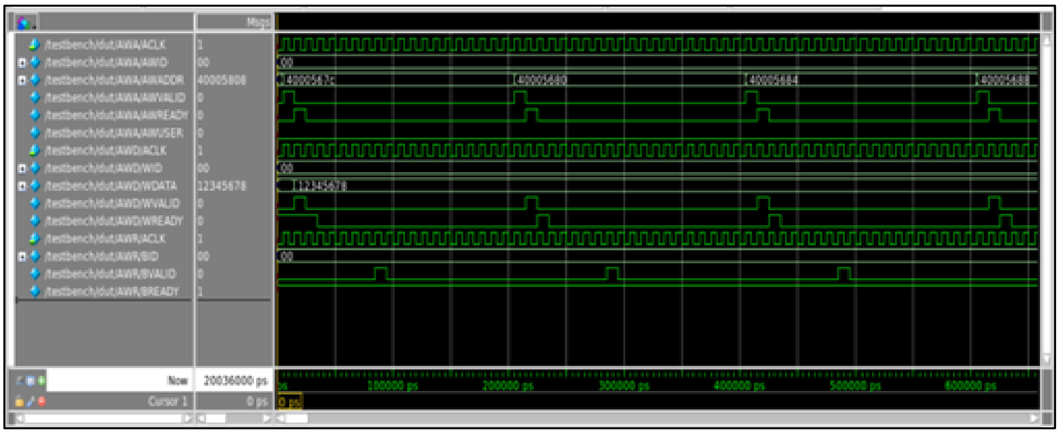

Msgs
/testbench/dut/AWA/ACLK 1
/testbench/dut/AWA/AWID 00
/testbench/dut/AWA/AWADDR 40005808
/testbench/dut/AWA/AWVALID 0
/testbench/dut/AWA/AWREADY 0
/testbench/dut/AWA/AWUSER 0
/testbench/dut/AWD/ACLK 1
/testbench/dut/AWD/WID 00
/testbench/dut/AWD/WDATA 12345678
/testbench/dut/AWD/WVALID 0
/testbench/dut/AWD/WREADY 0
/testbench/dut/AWR/ACLK 1
/testbench/dut/AWR/BID 00
/testbench/dut/AWR/BVALID 0
/testbench/dut/AWR/BREADY 1
00
4000567c
40005680
40005684
40005688
12345678
Now 20036000 ps
Cursor 1 0 ps
0 ps
100000 ps
200000 ps
300000 ps
400000 ps
500000 ps
600000 ps

Fig. 5: AXI Write Channel signals

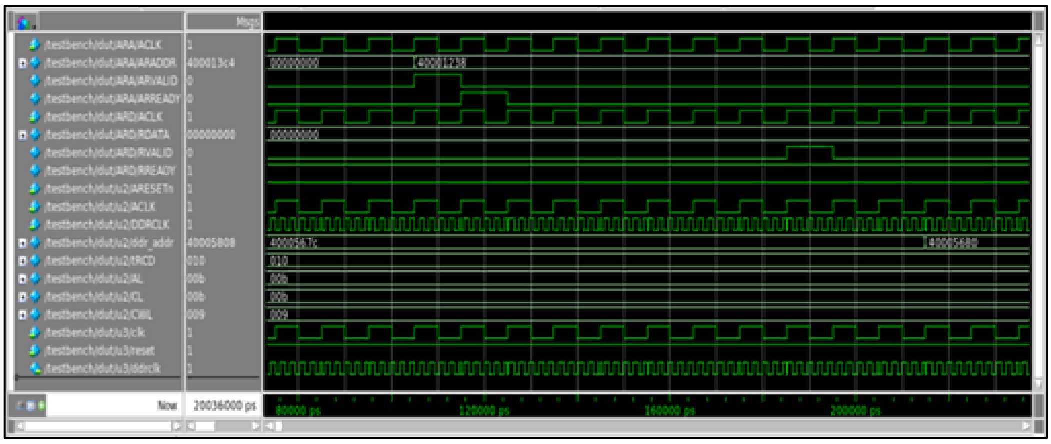

Msgs
/testbench/dut/ARA/ACLK 1
/testbench/dut/ARA/ARADDR 400013c4
/testbench/dut/ARA/ARVALID 0
/testbench/dut/ARA/ARREADY 0
/testbench/dut/ARD/ACLK 1
/testbench/dut/ARD/RDATA 00000000
/testbench/dut/ARD/RVALID 0
/testbench/dut/ARD/RREADY 1
/testbench/dut/u2/ARESETn 1
/testbench/dut/u2/ACLK 1
/testbench/dut/u2/DDRCLK 1
/testbench/dut/u2/ddr_addr 40005808
/testbench/dut/u2/tRCD 010
/testbench/dut/u2/AL 00b
/testbench/dut/u2/CL 00b
/testbench/dut/u2/CWL 009
/testbench/dut/u3/clk 1
/testbench/dut/u3/reset 1
/testbench/dut/u3/ddrclk 1
00000000
40001238
4000567c
40005680
Now 20036000 ps
80000 ps
120000 ps
160000 ps
200000 ps

Fig. 6: AXI Read Channel Signals

B. Latency Measurement:

Request is initiated at initiator and response is given by memory controller, responses may or may not be in order. So time taken to respond the request is known as latency [4]. Below Fig. 7 shows write latency measurement waveforms and Fig. 8 represents read latency waveforms, which is also observed using monitor and extracted during simulations by monitor.

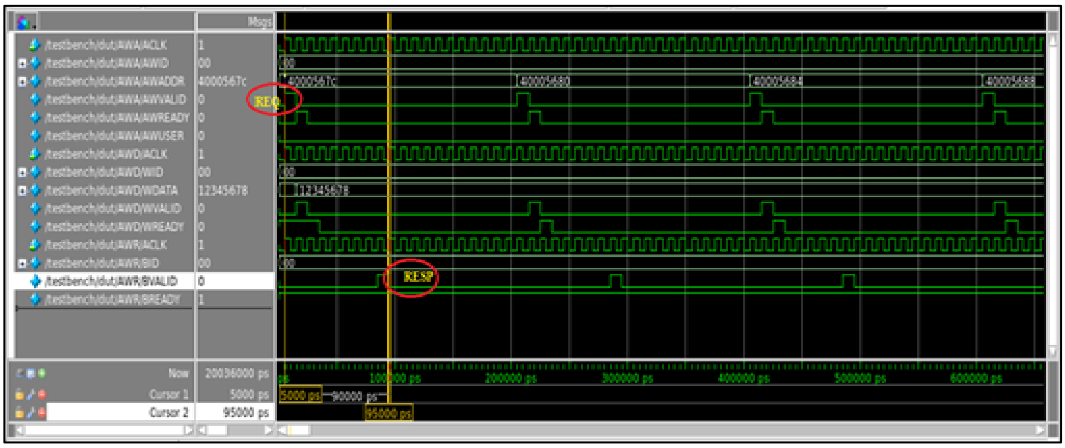

Msgs
/testbench/dut/AWA/ACLK 1
/testbench/dut/AWA/AWID 00
/testbench/dut/AWA/AWADDR 4000567c
/testbench/dut/AWA/AWVALID 0
/testbench/dut/AWA/AWREADY 0
/testbench/dut/AWA/AWUSER 0
/testbench/dut/AWD/ACLK 1
/testbench/dut/AWD/WID 00
/testbench/dut/AWD/WDATA 12345678
/testbench/dut/AWD/WVALID 0
/testbench/dut/AWD/WREADY 0
/testbench/dut/AWR/ACLK 1
/testbench/dut/AWR/BID 00
/testbench/dut/AWR/BVALID 0
/testbench/dut/AWR/BREADY 1
REQ
RESP
4000567c
40005680
40005684
40005688
12345678
Now 20036000 ps
Cursor 1 5000 ps
Cursor 2 95000 ps
100000 ps
200000 ps
300000 ps
400000 ps
500000 ps
600000 ps
5000 ps
90000 ps
95000 ps

Fig. 7: Write Latency

Write request is at 5000ps and response is coming at 95000ps from Fig. 7.

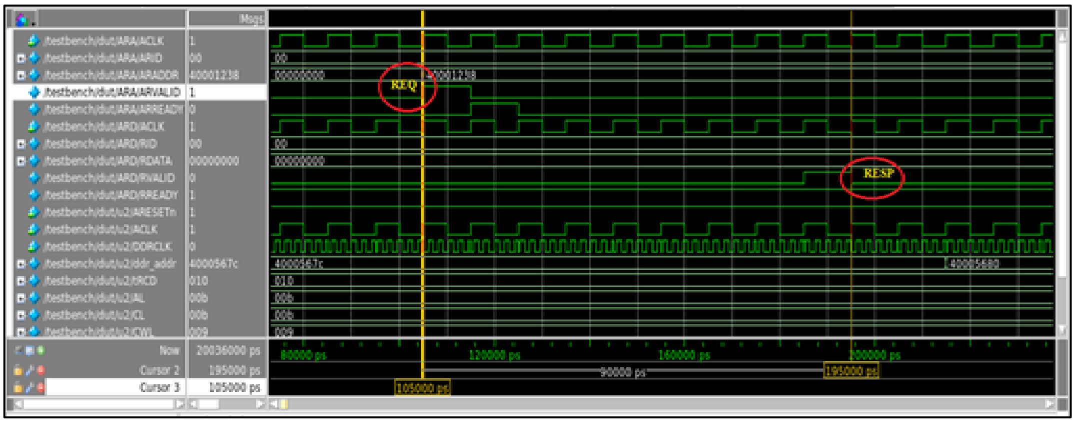

Msgs
/testbench/dut/ARA/ACLK 1
/testbench/dut/ARA/ARID 00
/testbench/dut/ARA/ARADDR 40001238
/testbench/dut/ARA/ARVALID 1
/testbench/dut/ARA/ARREADY 0
/testbench/dut/ARD/ACLK 1
/testbench/dut/ARD/RID 00
/testbench/dut/ARD/RDATA 00000000
/testbench/dut/ARD/RVALID 0
/testbench/dut/ARD/RREADY 1
/testbench/dut/u2/ARESETn 1
/testbench/dut/u2/ACLK 1
/testbench/dut/u2/DDRCLK 0
/testbench/dut/u2/ddr_addr 4000567c
/testbench/dut/u2/tRCD 010
/testbench/dut/u2/AL 00b
/testbench/dut/u2/CL 00b
/testbench/dut/u2/CWL 009
REQ
RESP
40001238
40005680
Now 20036000 ps
Cursor 2 195000 ps
Cursor 3 105000 ps
80000 ps
120000 ps
160000 ps
200000 ps
90000 ps
195000 ps
105000 ps

Fig. 8: Read Latency

As shown in Fig8. Read request is at 105000ps and response is coming at 195000ps.

Transactions		**Performance Parameters**			
		Read Bandwidth	***Write Bandwidth***	***Read Latency***	***Write Latency***
Non-Pipelined	No page open optimize	15Mbps	15Mbps	12	13
	Page open only when required	40Mbps	40Mbps	9	9
	Additive Latency(AL)=0	52Mbps	52Mbps	6	7
Pipelined	Pipelined transaction	66Mbps	66Mbps	4	4
Non-sequential	With ACLK=1000ns	110Mbps	110Mbps	3	3

Table 2: Transactions based Performance Parameters

VI. Conclusion

This paper exhibits a performance evaluation of DDR4 memory controller using UVM based verification environment, test-bench is built to assure that DDR4 memory controller can effectively access the memory transactions from the AXI4. AXI4 read, write transactions, latency and bandwidth measurements in data less simulation has been performed. DDR4 memory controller performance enhancement was realized from development of designs like page open strategy, pipelined transactions, scheduling algorithms for ideal latency and multiple bank group appearances given by DDR4 memory.

REFERENCES

[1] Khaled Salah Khaled Khalifa. Implementation and verification of a generic universal memory controller based on uvm. In 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era, pages 1–2. IEEE, 2015.

[2] Ou Guodong Dou Qiang Xie Lunguo Pan Guoteng, Luo Li. Design and implementation of a ddr3-based memory controller. In 2013 Third International Conference on Intelligent System Design and Engineering Applications, pages 540–543. IEEE,
2012.

[3] Md. Yeasin Arafath Md. Ashraful Islam and Md. Jahid Hasan. Design of ddr4 sdram controller. In 8th International Conference on Electrical and Computer Engineering 20-22 December, 2014, Dhaka, Bangladesh, pages 148–151. IEEE, 2014.

[4] Benny Akesson Sven Goossens, Karthik Chandrasekar and Kees Goossens. Power/performance trade-offs in real-time sdram command scheduling. In IEEE TRANSACTIONS ON COMPUTERS, VOL. 65, NO. 6, JUNE 2016, pages 1882–1895. IEEE, 2016.

[5] nam-po kim byeong min Yomy-Nam yun, jae-beom kim. Beyond uvm for practical soc verification. In IOSC 2011, pages 158–162. ISOOC, 2011.

[6] Xinan Wang Lingling Chai, Zheng Xie. A verification methodology for reusable test cases and coverage based on system verilog. In 2014 IEEE international conference on electronic devices and solid state circuits, pages 1–2. IEEE, 2014.

[7] Rolf Drechsler Marcio F. S. Oliveira, Finn Haedicke. The system verification methodology for advanced tlm verification. In CODES+ISSS12, October 7-12, 2012, Tampere, Finland, pages 312–322. ACM, 2012.

[8] K. Niranjan Reddy G Sai Divya. Verification of amba axi4 protocol using uvm. In International Journal of Electrical Electronics Computer Science Engineering, pages 25–29. IEEE, 2015.

[9] Sakthivel. SM G.Mahesh. Verification of memory transactions in axi protocol using system verilog approach. In IEEE ICCSP 2015 conference, pages 0860–0864. IEEE, 2015.

[10] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[11] Accellera, Universal verification methodology(uvm) 1.1 users guide, CA, May 2011.

[12] AMBA. AXI and ACE protocol specification.pdf.

[13] Micron. Ddr4 sdram, 2014.

[14] Accellera, IEEE standard for system Verilog [online]. Available: http://accellera.org/downloads/ieee.