International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 13 (2015)

Design and Verification of Memory Controller AXI for DDR4 Memories

R.R.Kalyan Venkatesh*[1], M.Harinath[*2], B.Murali Krishna[*3]

[1] *Student of VLSI Systems Research Group, Department of Electronics and Communication Engineering, K L University, AP-INDIA*
Email: kalyanavenkatesh495@gmail.com
[2] *Student of VLSI Systems Research Group, Department of Electronics and Communication Engineering, K L University, AP-INDIA*
Email: harinathmandhalapu@gmail.com
[3]*Asso. Professor VLSI Systems Research Group, Department of Electronics and Communication Engineering, KL University, AP-INDIA*
Email: bopanna.muralikrishna@gmail.com

ABSTRACT
AXI (Advanced extensible interface), the third generation of AMBA (Advanced Microcontroller Bus architecture) interface is targeted for high performance, high clock frequency designs. AXI is used in Memory controllers, and DMA controllers. AXI has many advantages over many other interfaces like SPI, APB. Ability to issue multiple outstanding addresses. UVM (Universal Verification Methodology) is the well proven standard methodology for verifying any integrated circuit designs. UVM environment consists of reusable verification components. UVM environment consists of UVC/agent, Scoreboard, and coverage Collector etc .UVC (UVM Verification Component) generates stimulus to the DUV (Design under Verification) and monitors the transaction which takes place at the DUV interface.UVC consists of a Driver, Monitor, and Sequencer. Driver is for driving/generating the inputs to the DUV and Monitor is to monitor the transaction taking place at the DUV interface .The tool used for this project is Modelsim 10.1c.

Keywords—AXI, AMBA, DDR4, UVM, DUV.

I. INTRODUCTION

As semiconductor technology improves, System-On-Chip (SoC) designs are becoming popular. A SoC platform usually consists of various design components dedicated to specified application domains. In order to ensure the functional correctness of a SoC, finding and fixing the design errors at early design phases is important in today's hardware development flows. The process of finding design errors is called "verification". However, as design complexity increases, experience shows that many bugs remain undetected even though considerable resources and time have been devoted to design verification, which can cause serious problems. Due to the importance of ensuring a design's functional correctness, a great deal of effort has been devoted to design verification in order to reduce verification effort and promote design quality. However, most existing direct test simulation-based verification techniques cannot guarantee sufficient coverage of the design, resulting in undetected bugs. Furthermore, they cannot accurately handle non-deterministic problems that are becoming more and more important nowadays. In order to overcome the limitation of direct test verification & random verification techniques, constraint random verification (CRV) to verify design correctness has been proposed as a promising alternative to direct test simulation. To improve design quality and reduce verification effort, we propose Coverage Driven Constraint Random Verification in this dissertation.

With advances in semiconductor technology, functional verification continues to remain one of the primary challenges in SoC designs today. As the statistics in industry surveys show, despite the fact that up to 70% of project resources have been devoted to functional verification, only 33% of SoC designs are correct on the first pass, and 75% of all design flaws are attributable to logic or functional bugs due to shortcomings in functional verification. The Coverage Driven Verification (CDV) combines automatic test generation, self-checking, test benches, and coverage metrics to significantly reduce the time spent verifying a design and reach the coverage goal. It track progress with functional coverage to ensure test plan criteria are met and also ensures corner cases are hit. System Verilog language provides additional flexibility for writing constraints. This method contributes to the overall verification methodology by ensuring well-defined properties that can guarantee the correctness in and between blocks. This improves modularity of verification and enables finding bugs in the design process when they are easier to understand and fix.

Verification is an integral part of any design. It starts parallel with the development of the design. For efficient development of design there need to be clear understanding between the design and verification engineers. In semiconductor industry about 70 % of resources are utilized for verification. In the past verification was done by individual engineers, the functionality of individual blocks, each verification component was decided by those verification engineers. Verification test bench was coded in their own style which made other verification engineers to understand their codes. To overcome this problem ACCLEERA developed a standard verification methodology called UVM (Universal Verification Methodology).Components which are implemented under UVM can be reusable, reducing the amount to purchase the new IP. UVC (UVM Verification Component)/ UVM Agent which is implemented using UVM can function according to any desired protocol (in this project it is AXI protocol).UVC

functions like a Bus Functional Model (BFM) which generates stimulus to Design under Verification (DUV). In addition to generating stimulus to the design under verification, UVC monitors the transactions which take place at interface of the DUV. UVC /UVM agent implemented in this project is AXI compliant and can be used any design which follows AXI protocol. Hence, UVC/BFM is reusable. AXI (Advanced Extensible Interface) is one of the AMBA (Advanced Microcontroller Bus Architecture) buses. It is 3rd generation AMBA bus. It has several advantages over other AMBA buses like AHB.AXI is also backward compatible with other AMBA buses like AHB and APB. Advanced Extensible Interface (AXI) is normally used in high frequency designs like memory controllers, DMA controllers. Here the section II describes about the AXI Channels and Architecture and section III represents the UVM verification environment. Finally Simulation results & Conclusion of this paper described in section IV, V.

II. AXI CHANNELS & ARCHITECTURE

This paper discusses about the advantages of AXI protocol, different channels present in AXI and their corresponding signals, handshaking process between signals AXI has 5 different channels viz. Write address channel, Write data channel, write response channel, Read address channel and Read data channel. Figure 2.1 shows the channel architecture of writes and figure 2.2 shows the channel architecture of reads.

Figure 2.1: Channel architecture of writes

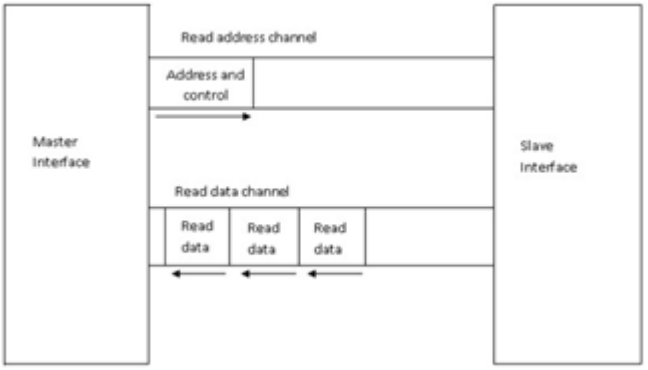


Figure 2.2: Channel architecture of reads

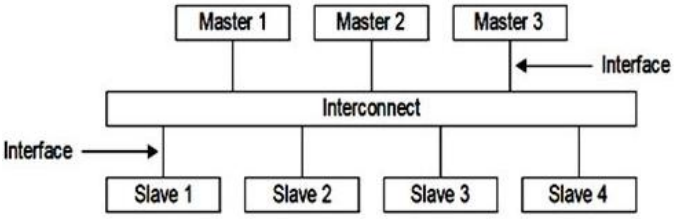


Figure 2.3: Interconnect for AXI

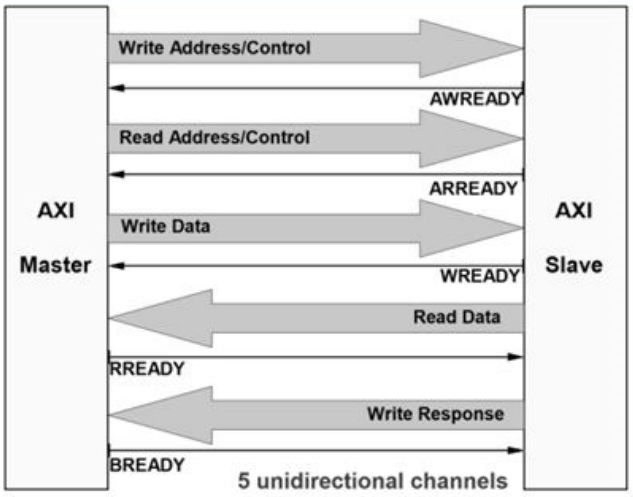


Figure 2.4: AXI Master Slave Communication

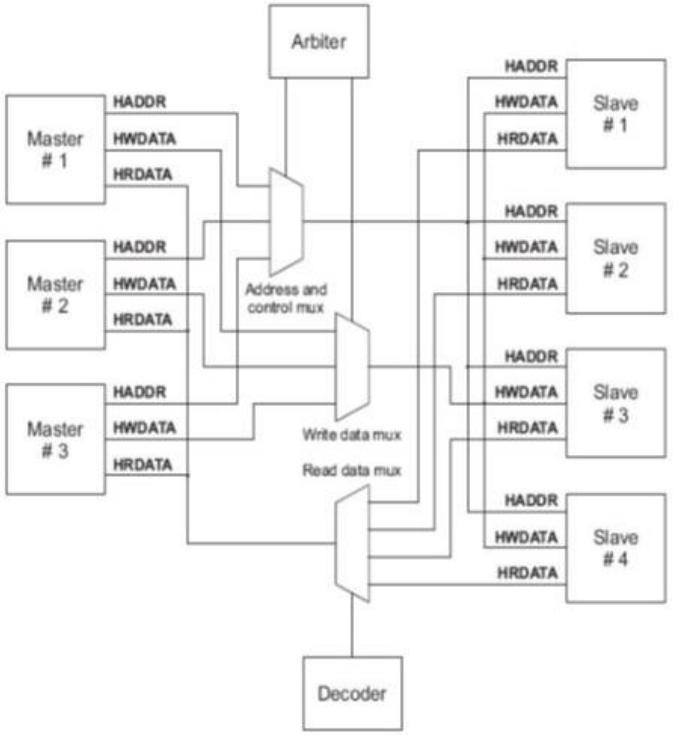


Figure 2.5: AXI Multi layer Master Slave Communication

WRITE ADDRESS CHANNEL

Write address channel carries address and required control information for a transaction. The AXI protocol supports the following

- Variable-length bursts, from 1 to 16 data transfers in a burst.
- Bursts with a transfer size 8-1024 bits.
- Different types of bursts, viz. wrapping, incrementing and non-incrementing bursts.
- Atomic operation, using exclusive or locked accesses
- Secure and privileged access.

Table 2.1: Write address channel signals and description

Signal Name	**Source**	**Description**	
AWID[3:0]	Master	Write Address ID, for a transaction	
AWADDR[31:0]	Master	Write Address.	
AWLEN[3:0]	Master	Burst length. Burst length gives the exact number of transfers in a burst.	
		AWLEN	**No. of Data transfers**
		0000	1
		0001	2
		0010	3
		“	“
		1111	16
AWSIZE[2:0]	Master	Burst size. This indicates the size of each transfer in a burst	
		AWSIZE	**Bytes in transfer**
		000	1
		001	2
		010	4
		011	8
		100	16
		101	32
		110	64
		111	128
AWBURST[1:0]	Master	Burst type. Depending on the type of the burst, slave will calculate the remaining address.	
		AWBURST	**Type of burst**
		00	Fixed
		01	Incrementing
		10	Non-Incrementing
		11	Reserved
AWLOCK[1:0]	Master	Lock type. This signal indicates the atomic characteristics of the transfer.	
		AWLOCK	**Access type**
		00	Normal access
		01	Exclusive access
		10	Locked access
		11	Reserved
AWPROT[2:0]	Master	Protection type.	
		AWPROT	**Protection level**
		1	Privileged access
		0	Normal access
		1	Non-secure access
		0	Secure access
		1	instruction access
		0	data access
AWC ACHE[3:0]	Master	Cache type.	
		AWC ACHE	**Transaction attribute**
		0000	Non-cacheable & non-Bufferable
		0001	Bufferable only
		0010	Cacheable, do not allocate
		0011	Cacheable & Bufferable, but do not allocate
		0100	Reserved
		0101	Reserved
		0110	Cacheable write through, allocate on read only
		0111	Cacheable write back, allocate on read only
		1000	Reserved
		1001	Reserved
		1010	Cacheable write through, allocate on write only
		1011	Cacheable write back, allocate on write only
		1100	Reserved
		1101	Reserved
		1110	Cacheable write through, allocate on both read & writes
		1111	Cacheable write back, allocate on both read & Writes
AW VALID	Master	Write address valid. This signal indicates that valid write address	

		and control information are available: 0- address and control information not available 1- address and control information available
AWREADY	Slave	Write address ready. This signal indicates if the slave is ready to accept the address and control information: 0- slave not ready 1- slave ready

READ ADDRESS CHANNEL

Read address channel carries address and required control information for a transaction. The AXI protocol supports the following

- Variable-length bursts, from 1 to 16 data transfers in a burst
- Bursts with a transfer size 8-1024 bits
- Different types of bursts, viz. wrapping, incrementing and non-incrementing bursts
- Atomic operation, using exclusive or locked accesses
- Secure and privileged access.

Table 2.2: READ ADDRESS CHANNEL SIGNALS AND DESCRIPTIONS

Signal Name	Source	Description	
ARID[3:0]	Master	Read Address ID, for a transaction	
ARADDR[31:0]	Master	Read Address.	
ARLEN[3:0]	Master	Burst length. Burst length gives the exact number of transfers in a burst.	
		ARLEN	No. of Data transfers
		0000	1
		0001	2
		0010	3
		“	“
		1111	16
ARSIZE[2:0]	Master	Burst size. This indicates the size of each transfer in a Burst	
		ARSIZE	Bytes in transfer
		000	1
		001	2
		010	4
		011	8
		100	16
		101	32
		110	64
		111	128
ARBURST[1:0]	Master	Burst type. Depending on the type of the burst, slave will calculate the remaining address.	
		ARBURST	Type of burst
		00	Fixed
		01	Incrementing
		10	Non-incrementing
		11	Reserved
ARLOCK[1:0]	Master	Lock type. This signal indicates the atomic characteristics of the transfer.	
		ARLOCK	Access type
		00	Normal access
		01	Exclusive access
		10	Locked access
		11	Reserved
ARPROT[2:0]	Master	Protection type.	
		ARPROT[2:0]	Protection level
		1	Privileged access
		0	Normal access
		1	Non-secure access
		0	Secure access
		1	instruction access
		0	data access
ARCACHE[3:0]	Master	Cache type.	
		ARCACHE	**Transaction attribute**
		0000	Non-cacheable & non-bufferable
		0001	Bufferable only
		0010	Cacheable, do not allocate
		0011	Cacheable & bufferable, but do not allocate
		0100	Reserved
		0101	Reserved
		0110	Cacheable write through, allocate on read Only
		0111	Cacheable write back, allocate on read only
		1000	Reserved
		1001	Reserved
		1010	Cacheable write through, allocate on write Only
		1011	Cacheable write back, allocate on write Only
		1100	Reserved
		1101	Reserved
		1110	Cacheable write through, allocate on both read & writes
		1111	Cacheable write back, allocate on both read & writes

ARVALID	Master	Read address valid. This signal indicates that valid write address and control information are available: 0- address and control information not available 1- address and control information available
ARREADY	Slave	Read address ready. This signal indicates if the slave is ready to accept the address and control information: 0- slave not ready 1- slave ready

III. UVM VERIFICATION COMPONENT (UVC)

This gives a description about what UVC is, different components present inside UVC, role of UVC in the UVM environment and communication inside the environment.

UVM

Universal Verification Methodology (UVM) is the standard methodology for verifying integrated circuits or complex designs, which was developed by ACCLERA. In the past verification was done by individual engineers, the functionality of individual blocks, each verification component was decided by those verification engineers. Verification test bench was coded in their own style which made other verification engineers to understand their codes .To overcome this problem ACCLEERA developed a standard verification methodology called UVM (Universal Verification Methodology). UVM is derived from industry proven methodologies like open verification methodology (OVM) and Verification methodology manual (VMM), and it also backward compatible with OVM and VMM. Components which are implemented under UVM are reusable. Hence UVM promotes reusability.

UVM ENVIORNMENT

Figure 3.1 shows the entire UVM Environment. The big dashed line box is the UVM environment. The UVM environment consists of AXI UVC/AXI Agent, Scoreboard, and Coverage Collector. Here LPDDR2MC is our DUV considered. The Small dashed line box indicates the UVC/Agent. Coverage Collector checks if all the design specification is covered, Scoreboard is used for data integrity check and memory monitor will check the transactions on the interfaces of the DUV and collects the transactions whenever observed on the interface. The communication within the environment takes place through TLM (Transaction Level Modeling) ports.

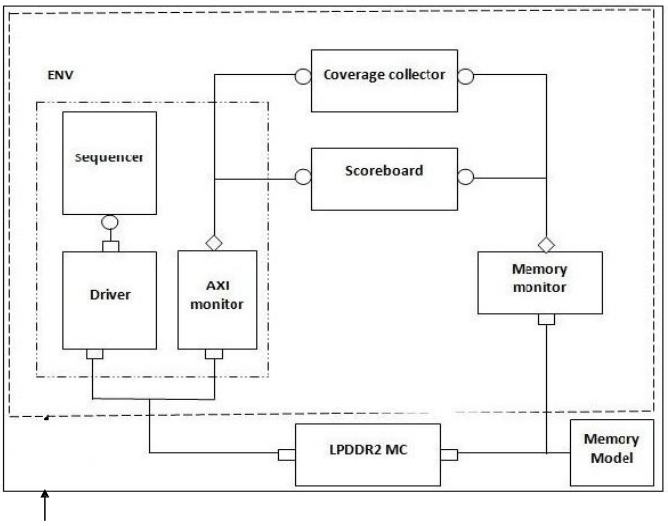


Figure 3.1 UVM Environment

UVM VERIFICATION COMPONENT (UVC)

Figure 3.2 shows the UVM Verification Component/Agent. Every Verification component has a consistent architecture, which has s sequencer, driver and a monitor. The communication within the agent is through TLM ports.

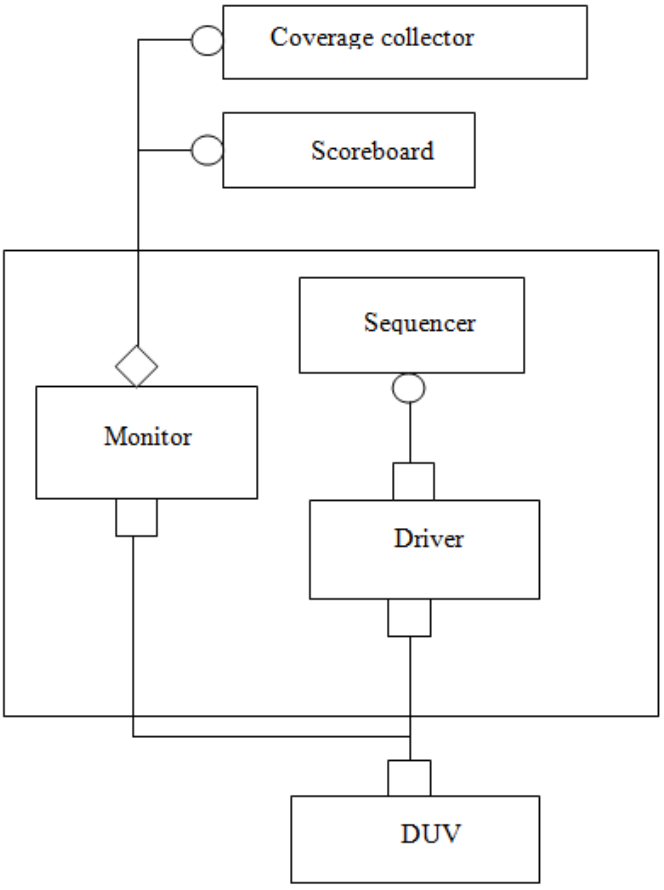


Figure 3.2 UVC/ AGENT

The UVC implemented in this project is compliant to AXI protocol. The UVC will generate stimulus according to AXI protocol.

SEQUENCE ITEM (DATA ITEM):
Sequence item is extended from uvm_sequence_item base class.
Data Item: The uvm_sequence_item base class includes the m_sequence_id field, allowing sequence items to be correlated to the sequence that generated them originally.

MONITOR:
The monitor observes pin level activity and converts its observations into sequence_items which are sent to components such as scoreboard, coverage collector.

CONFIG:
One of the important blocks in Agent is the config block. It is container block which decides if the agent should be active or passive. Configuration object contains an active bit to decide if the agent should active or passive. An Active agent consists of sequencer, driver and monitor as shown in the figure. Active agent generates and monitors the transaction which takes place on the DUV interface. A Passive Agent consists of only monitor which can only monitor the transactions which takes place at DUV interface.

AGENT:
An agent comprises of Sequencer, Driver and Monitor. An Agent is capable of generating and driving the stimulus to the DUV. An agent can be either active or passive.

TRANSACTION LEVEL MODELING (TLM)
The different TLM ports used in the agent are export, import and analysis port. The small square box which is connected to the driver is port, the small circle which is connected to the sequencer is export and the diamond figure which is connected to the monitor is analysis port.

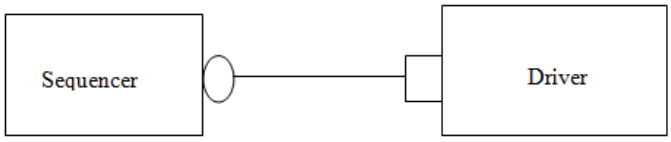


Figure 3.3: TLM communication between Sequencer and Driver

The communication between the sequencer and the driver within the agent is via TLM ports, viz. export and port as shown in the figure 3.3.The circle in figure represents the export and square box represents the port. Sequencer will the data items to the driver, only when the driver requests for the next data item The driver can also send a response back to the sequencer, which is optional. Sequence items (data items) are passed between the sequencer and driver; hence these are called as sequence_item_port and sequence_item_export. The driver will receive data item only when it requests for the data item from sequencer issuing a command called get_next_item. Driver will also indicate to the sequencer that the transaction of the particular data_item is complete by sending a command item_done. The connection between sequencer and driver is via sequencer port connect (driver.export) command. The connection between a port and export should be one-to-one connection only.

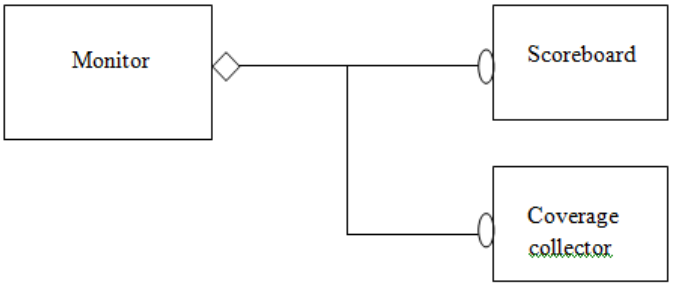


Figure 3.4: TLM communication between monitor and other analysis components

Figure 3.4 shows the TLM communication between monitor and other verification components present in the UVM environment such as coverage collector or scoreboard. The diamond shaped box in the above figure represents an analysis port. If nothing is connected, the write () call simply returns. Thus, an analysis port may be connected to zero, one, or many analysis exports, but the operation of the component that writes to the analysis port does not depend on the number of exports connected. Because write () is a void function, the call will always complete in the same delta cycle, regardless of how many components (for example, scoreboards, coverage collectors, and so on) are connected. In the environment, the analysis port gets connected to the analysis export of the desired components, such as coverage collectors and scoreboards. It is up to each component connected to an analysis port to provide an implementation of write () via an analysis_export. TLM promotes reusability as they have the same interface. It maximizes reuse and minimizes the time and effort compared to using mailboxes.

HOST UVC

In this paper, host Master which is the 33rd master will configure the registers according to different frequencies as per JESD209-2F specifications. Even this Host master will follow AXI protocol. In this project Host UVC has only Write address channel and write data channel. Host UVC also has sequence items, sequence, sequencer, driver and monitor. In the UVC of the host master, sequence items are constrained randomized according to the values in the JESD209-2F. There are three sets of registers used in this project Mode registers, Timing Registers and command configuration registers. All these registers are 32 bit register. Host UVC will configure the registers according to JESD209-2F specifications. The design specifications calculated according to JESD209-2F for 400 MHz for the Host UVC to configure is given below.

MODE REGISTERS

TABLE 3.1: Mode Register 1

7	6	5	4	3	2	1	0
nWR (for AP)			WC	BT	BL		

TABLE 3.2: Mode Register 1 Specification

SL No.	Mode Register 1 Specifications
1	The value of burst length BL in mode register 1 can be 4,8,16
2	The value of burst type BT in mode register 1 should be 0
3	The value of wrap type wc in mode register 1 should be 0
4	The value of nwr in mode register 1 should be 1

TABLE 3.3: Mode Register 2

7	6	5	4	3	2	1	0
RFU				RL and WL			

TABLE 3.4: Mode Register 2 Specifications

SL No.	Mode Register 2 Specifications
1	The value of rlwl which indicates the memory frequency in mode register 2 should be 1 for 200Mhz

TABLE 3.5: Mode Register 3

7	6	5	4	3	2	1	0
RFU				DS			

TABLE 3.6: MODE REGISTER 4

SL No.	Mode Register 3 Specifications
1	The value of ds which is the i/o configuration in mode register 3 should be 2

TIMING REGISTER

Timing register will store the timing values as per the memory specifications.

TABLE 3.7: REFRESH REGISTER

21-31	14-20	0-13
.....	tREFCab (Refresh cycle time for all banks)	tREFI (Avg. Time between Refresh commands)

TALE 3.8: REFRESH REGISTER SPECIFICATIONS

SL No.	Refresh Register Specifications
1	The value of trefi in refresh register should be 3120 for 400Mhz
2	The value of trefcab in refresh register should be 52 for 400Mhz

TABLE 3.9: ACTIVATE REGISTER

30-31	25-29	20-24	16-19	12-15	8-11	5-7	0-4
...	tFAW (Four bank activate window)	tRAS (Row active time)	tRPpb (Row precharge time per bank)	tRPab (Row precharge time all bank)	tRCD (RAS to CAS delay)	tRRD (Activate bank A to Activate bank B)	tRC (Activate to Activate command period)

TABLE 3.10: ACTIVATE REGISTER SPECIFICATIONS

SL No.	Activate Register Specifications
1	The value of trc in activate register should be 24
2	The value of trrd in activate register should be 4
3	The value of trcd in activate register should be 7
4	The value of trpab in activate register should be8
5	The value of trppb in activate register should be 7
6	The value of tras in activate register should be 17
7	The value of tfaw in activate register should be 20

This register indicates the time interval between the power down mode and self refresh.

TABLE 3.11: POWER DOWN REGISTER

21-31	3-20	0-2
.....	tDPD(Minimum Deep power down time)	tXP (Exit power down)

TABLE 3.12: POWER DOWN REGISTER SPECIFICATIONS

SL No.	Power Down Register Specifications
1	The value of txp in power down register should be 3
2	The value of tdpd in power down register should be 800000

This register indicates the number of refresh postpones allowed.

TABLE 3.13: POSTPONE REFRESH REGISTER

13-31	0-12
……	tPref

TABLE 3.14: POSTPONE REFRESH REGISTER SPECIFICATIONS

SL No.	Postpone Register Specifications
1	The value of tpref in postpone register should be 4055

COMMAND CONFIGURATION REGISTER

This register specifies device initialization of the memory. After configuring all the above mentioned registers, Host UVC will set 0^{th} bit of this register to 1 indicating device initialization of the memory is done.

MEMORY MONITOR

A memory monitor is another verification component in verification environment, similar to the AXI monitor, which extracts the signal information from the interface and translates it into transaction item. The memory monitor implemented in this project monitors all the transactions on the memory interface and passes these transaction signals to the verification components such as scoreboard and coverage collector via TLM ports. Memory interface carries all the signal information going from memory controller to memory.

COVERAGE COLLECTOR

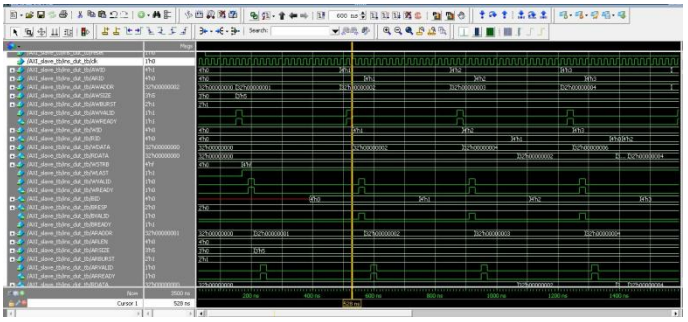
Coverage collector checks if all the design specifications are covered. Functional coverage is measured in coverage collector. It is a user defined metric that measures how much of the design specification, as enumerated by the features in test plan. It is based on the design specifications and it will indicate when to stop the verification and gauges the progress of the verification.

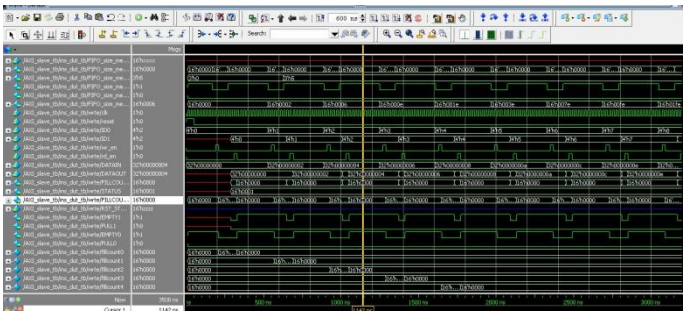
SCOREBOARD

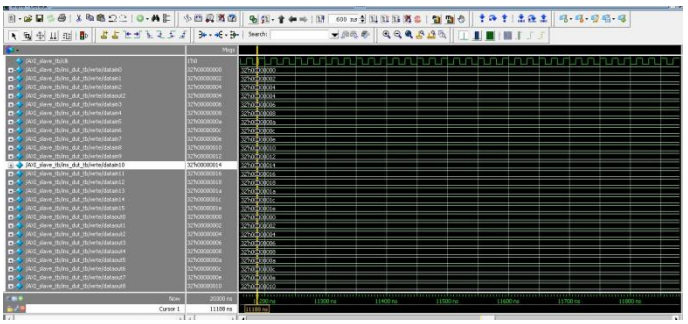
A crucial element of a self-checking environment is a scoreboard. Scoreboard verifies the proper operation of your design at a functional level. The responsibility of a scoreboard varies greatly depending on the implementation. Here it checks if the data is written by the master matches with data written in the memory location.

IV. SIMULATION RESULTS

Here the simulation results for the verification of the memory controller of AXI & DDR4 are given below.







V. CONCLUSION

The UVC implemented is configurable to any kind of AXI interfaces.UVC implemented supports 2 multiple outstanding transfers. Driver in the AXI transfers data according to the AXI protocol specifications. All the 5 channels of the AXI are implementing this UVC.AXI is a high speed bus, so AXI interface can be used in Memory controllers, DMA controllers, the UVC implementing in this paper can be used for those.

REFERENCES

[1] AMBA AXI Protocol V1.0
[2] UVM Users guide v1.1
[3] UVM Cookbook
[4] System Verilog LRM IEEE 1800-2005
[5] JESD209 2F specifications.
[6] Easier UVM for Functional Verification by Mainstream Users Updated and Extended for UVM 1.0 by John Aynsley.
[7] Verification of AMBA Bus Model Using System Verilog by Han Ke,Deng, zhongliang and Shu Qiong.
[8] Monitors, Monitors Everywhere-Who is Monitoring the Monitors by Rich Edelman and Raghu Ardeishar.
[9] If System Verilog Is SO Good, Why DO We Need the UVM? By Jonathan Bromley
[10] www.verificationacademy.com.