International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 07 | July-2016 www.irjet.net p-ISSN: 2395-0072

UVM BASED TEST BENCH TO VERIFY AMBA AXI4 SLAVE PROTOCOL

Smitha A P1, Ashwini S2
[1]M.Tech VLSI Design and Embedded Systems, ECE Dept.
[2]Assistant Professor, ECE Dept. NCET, Bengaluru, India.

Abstract - *The increasing amount of logic that can be placed onto a single silicon die is driving the development of highly integrated SoC designs. An important feature for any of the SoC is based on how they interconnect. AMBA protocols are today the de facto standard SoC bus because they are well documented and can be used without royalties. The AMBA AXI protocol supports high performance, high-frequency system designs. It is suitable for high-bandwidth, low-latency designs and provides high frequency operation without using complex bridges. It provides flexibility in the implementation of interconnect architectures and is backward-compatible with existing AHB and APB interfaces. This paper is aimed to design transaction between one master and one slave in Verilog and a burst type transaction (INCR) of AMBA AXI4 Slave Interface is verified using Universal Verification Methodology (UVM) and simulation results are shown in cadence Incisive Enterprise Simulator (IES).*

Keywords: System-on-a-Chip (SoC), Intellectual property (IP) Advanced Extensible Interface (AXI), ARM (Advanced RISC Machines) Advanced Peripheral Bus (APB), AMBA High performance Bus (AHB), Advanced Micro Controller Bus Architecture (AMBA), Universal Verification Methodology (UVM),Design under test (DUT), coverage driven verification (CDV).

1. INTRODUCTION

Integrated circuits have entered the era of System-on-a-Chip (SoC).SoC refers to integration of more different function IP's. The designers simply integrate their owned IPs with third party IPs into the SoC to significantly reduce design cycles. Now the common problem is communication among IP's. The interfaces to these IP's differs from company to company. To speed up SoC integration and promote IP reusability, many bus-based communication architecture standards have emerged over the past several years. Some of the popular standards include ARM Microcontroller Bus Architecture (AMBA) versions of 2.0, 3.0, and 4.0, IBM Core Connect, STMicroelectronics STBus, Sonics SMARRT Interconnect, Open Cores Wishbone, and Altera Avalon. However, the main issue is that to ensure efficiently the IP functionality, which works properly after integrating to the corresponding bus architecture. The AMBA AXI4 protocol is a standard bus protocol and most of the semiconductor companies design supports AXI4 bus interface. AXI4 protocol is a complex protocol because of its ultra-high-performance. On current projects, verification engineers are maximum compared to designers, with the ratio reaching 2 or 3 to one for the most complex designs. Therefore an efficient verification environment is needed. Verification of such a complex protocol is challenging. This can be easily verified using the UVM. This verification environment can be reused for other IPs also. UVM is a complete verification methodology that codifies the best practices for development of verification environments targeted at verifying large gate-count, IP-based SoC's. It is mainly used to write a test bench for all those designs which are modelled in Verilog, VHDL, and System C. It has a System Verilog class library which helps to achieve the reusability. It supports constrained random coverage driven verification. CDV is a combination of automatically generation of test benches, self-checking of test benches and coverage metrics.

2. AMBA AXI4 ARCHITECTURE

AXI4 is a part of the Advanced Microcontroller Bus Architecture (AMBA) which is developed by ARM Company. The AMBA AXI4 is used for high performance, high frequency system designs. It also supports high bandwidth and low latency designs. It allows high frequency operation without use of complex bridges. AMBA AXI4 supports data transfers up to 256 beats and unaligned data transfers using byte strobes. In AMBA AXI4 system, 16 masters and 16 slaves are interfaced. Each master and slave has their own 4 bit ID tags. It satisfies the interface requirements of all components. It supports memory controllers which has high latency access. It is flexible in the implementation of many interconnects. It is compatible with AHB and APB interfaces.

Main features of the AXI are:

1. Separate address/control and data phases
2. Unaligned data transfer using byte strobes
3. It supports for burst based mode of transactions with the issue of start address
4. Separate read data channels and write data channels
5. Provides multiple outstanding addresses
6. Supports for out-of-order transaction completion
7. Provides easy addition of registers for time closure

2.1AXI4 transaction channels

The AXI protocol supports burst based transactions. AXI supports five independent transaction channels. Figure 1 shows read and write transaction using channels.

- Read address channel

- Read data channel
- Write address channel
- Write data channel
- Write response channel

Figure 1. AXI4 channel architecture of read and write [3]

Read and write address channel: An address channel read/write carries the control information that reports the nature of data to be transferred. The transfer of data takes place between master and slave either through write data channel or read data channel.

Read data channel: It conveys both read data and read response information from slave to master. The data may be of size 8,16,32,64,128,256,512 or 1024 bits wide. A read response signal indicates the completion of read transaction.

Write data channel: It is used to transfer data from master to slave. Data size can be 8,16,32,64,128,256,512 or 1024 bits wide. A byte lane strobe signal is generated for every eight data bits to indicate which bytes of the data are valid.

Write Response channel: It provides a way for the slave to respond to write transactions. All write transactions require completion of signaling on the write response channel.

2.2. AXI4 handshake process

In AXI4 protocol, every transfer is done using hand shake mechanism. Each channel uses the same VALID/READY handshake to transfer control and data information. This two way flow control mechanism enables both the master and slave to control the rate at which the data and control information moves. The source generates the VALID signal to indicate when the data or control information is available. The destination generates the READY signal to indicate that it accepts the data or control information. Transfer occurs only when both the VALID and READY signals are HIGH. There must be no combinatorial paths between input and output signals on both master and slave interfaces.

2.3AXI4 burst operation

The AXI protocol defines three burst types:

*FIXED burst***:** In a fixed burst, the address is the same for every transfer in the burst. This burst type is used for repeated accesses to the same location such as when loading or emptying a FIFO.

INCR burst: In an incrementing burst, the address for each transfer in the burst is an increment of the address for the previous transfer. The increment value depends on the size of the transfer. For example, the address for each transfer in a burst with a size of four bytes is the previous address plus four. This burst type is used for accesses to normal sequential memory.

WRAP burst: A wrapping burst is similar to an incrementing burst except that the address wraps around to a lower address if an upper address limit is reached.

2.4. AXI4 basic transaction

Read burst: Figure 2 shows a read burst of four transfers. Here, the master drives the address, and the slave accepts it one cycle later. After the address appears on the address bus, the data transfer occurs on the read data channel. The slave keeps the VALID signal low until the read data is available. For the final data transfer of the burst, the slave asserts the RLAST signal to show that the last data item is being transferred.

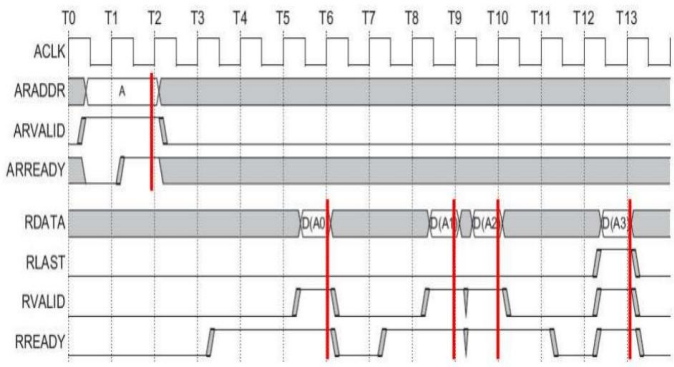

Figure 2.Read address and data burst [3]

*Write burst***:** Figure 3 shows a write transaction. The process starts when the master sends an address and control information on the write address channel. The master then sends each item of write data over the write data channel. When the master sends the last data item, the WLAST signal goes high. When the slave has accepted all the data items, it

drives a write response back to the master to indicate that the write transaction is complete.

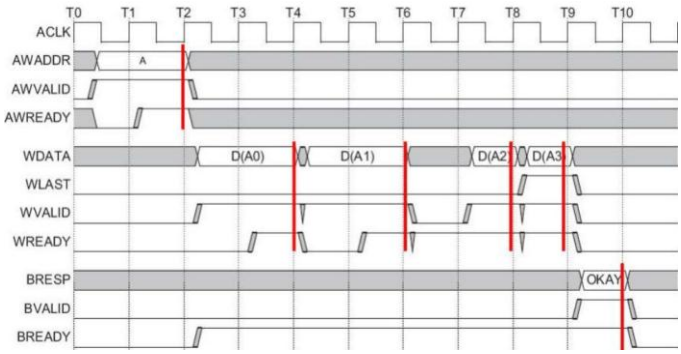

Figure 3.Write address and data burst [3]

TABLE 1: Signal descriptions of AMBA AXI4 protocol.

Signal	Source: master/ slave	Input/ Output	Description
Aclk	Global	Input	Global clock signal.
AResetn	Global	Input	Global reset signal
AWID[3:0]	Master	Input	Write address ID.
AWADDR[31:0]	Master	Input	Write address.
AWLEN[3:0]	Master	Input	Write burst length.
AWSIZE[2:0]	Master	Input	Write burst size.
AWBURST[1:0]	Master	Input	Write burst type.
AWLOCK[1:0]	Master	Input	Write lock type.
AWCACHE[3:0]	Master	Input	Write cache type.
AWPROT[2:0]	Master	Input	Write protection type.
WDATA[31:0]	Master	Input	Write data.
ARID[3:0]	Master	Input	Read address ID.
ARADDR[31:0]	Master	Input	Read address.
ARLEN[3:0]	Master	Input	Read Burst length.
ARSIZE[2:0]	Master	Input	Read Burst size.
ARLOCK[1:0]	Master	Input	Read Lock type.
ARCACHE[3:0]	Master	Input	Read Cache type.
ARPROT[2:0]	Master	Input	Read Protection type.
RDATA[31:0]	Master	Input	Read data.
WLAST	Master	Input	Write last.
RLAST	Slave	Output	Read last.
AWVALID	Master	Output	Write address valid.
AWREADY	Slave	Output	Write address ready.
WVALID	Master	Output	Write valid.
RAVLID	Slave	Output	Read valid.
WREADY	Slave	Output	Write ready.
BID[3:0]	Slave	Output	Write Response ID.
RID[3:0]	Slave	Output	Read response ID.
BRESP[1:0]	Slave	Output	Write response.
RRESP[1:0]	Slave	Output	Read response.
BVALID	Slave	Output	Write response valid.
BREADY	Master	Output	Response ready.
RVALID	Slave	Output	Read valid.

3. ARCHITECTURE OF UVM TEST BENCH

Universal Verification Methodology (UVM) is used for functional verification of hardware or any system. Hardware or any system is written in Verilog, System c, VHDL or System Verilog at any abstraction level. Abstraction level may be behavioral or gate level or register transfer level. UVM is a simulation based verification methodology. UVM can also be used along with Assertion based verification or emulation. UVM is a constrained random coverage driven verification (CDV).CDV is a combination of automatically generation of test benches, self-checking of test benches and coverage metrics. The main aim of CDV is

- Generation of random test vectors
- Supports thorough verification
- Early notification of error to minimise debugging time

CDV flow is different from directed testing flow. In CDV, verification goals are set with organized plan. Then creation of test benches that generates legal stimulus and sends it to the DUT.

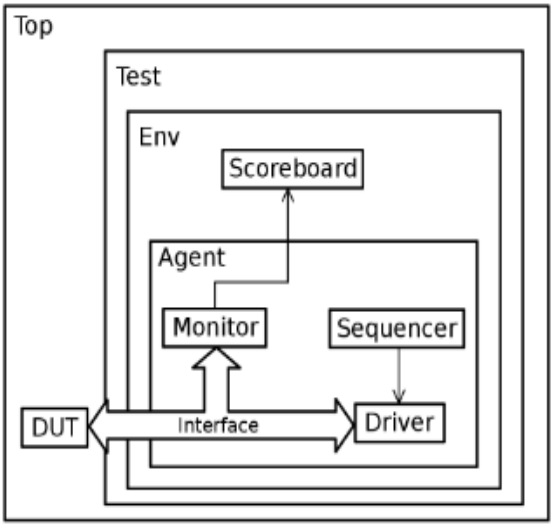

Figure 4. UVM environment [10]

In order to test its functionality of DUT, there is a need of interaction with DUT. A block by name sequencer generates sequences of bits to be transmitted to the DUT. Usually sequencers are not aware of the communication bus. Sequencer permits the sequences of data to the block by name driver. Driver takes care of the communication bus. Driver feeds the data to the DUT generated from the sequencer. But it does not do any validation of responses. Monitor block listens to the broadcast between the driver and the DUT. It determines the responses from the DUT. Monitor samples the input and output of the DUT. Monitor make a prediction of the expected result and send the

prediction and result of the DUT to scoreboard block in order to compare and evaluate. All these blocks constitute a system which is used for verification.

Generally sequencer block, driver block and monitor block comprise an agent. An agent and a scoreboard combine an environment. All these blocks are controlled by greater block test. Test block controls all the blocks and sub blocks of the test-bench. The UVM blocks are shown in figure 4. By manipulating a small number of lines in code, blocks can be added, removed and can be overruled the blocks in the test bench and different environment can be built without rewriting the test benches again.

3.1. UVM Components

Top block and interface

In a normal project, the development of the DUT is done separately from the development of the testbench, so there are two components that connects both of them:

- The top block of the testbench
- A virtual interface

The top block will create instances of the DUT and of the testbench. Virtual interface will act as a bridge between them. The interface is a module that holds all the signals of the DUT. The monitor, the driver and the DUT are all going to be connected to this module.

Sequencer and Sequences

Sequences signifies the input to the DUT, such as instructions, networking packets and bus transactions. A sequencer is a new stimulus generator that pedals the sequences which are delivered to a driver for the purpose of execution. Sequences are well-arranged assembly of transactions. They shape transactions as per specification and produce collection of them. Sequences main job is creating multiple transactions. After creating those transactions, the sequencer takes them to the driver.

Driver

The driver is a dynamic block that imitates logic and is used to drive the DUT. A driver continually receives a transaction from sequencer and drives the signal to the DUT by sampling.

Monitor

A monitor is the passive element of the verification environment and is independent to an application. It scans the DUT signal to and from the interface without driving them. It assembles the pin information in the form of a packet and then transfers it to scoreboard and test verification environment for coverage information.

Agent

An agent is a container which holds drivers, sequencer and monitors. Monitors, sequencers and drivers can be used independently. Verification components can contain more than one agent. Some agents (master) initiate transactions for the DUT, other agents (slave) respond to transactions. Agents can be configured to act as passive or active. Active agents initiate transactions. Passive agents observes activity of DUT.

Scoreboard

Scoreboard is built to check the response from the DUT against the expected response. It is done by comparing them to the Reference Model. It keeps the track of how many times the response matched and how many times it failed.

Environment

The environment is in top of the UVM verification environment. It instantiates the scoreboard and an agent and joins them. It consists of one or more agents, and a bus monitor. The environment enables to modify the topology and performance to create reusable, flexible, extendable verification environment.

Test

Test block is a top level block in UVM. It has 2 purpose

- Create the environment block
- Connect the sequencer to the sequence

By specifying in the test class which sequence will be going to be generated in the sequencer, can be easily change the kind of data transmitted to the DUT.

3.2. The UVM class library hierarchy

UVM component can be replaced easily without alteration of entire test-bench. This is because of the classes and objects. UVM blocks are represented as objects which are denoted as objects which are derived from existing classes. A tree structure of most vital classes of UVM shown in figure 5. UVM library has a set of base classes and services that enable the design of segmental, mountable, reusable verification environment

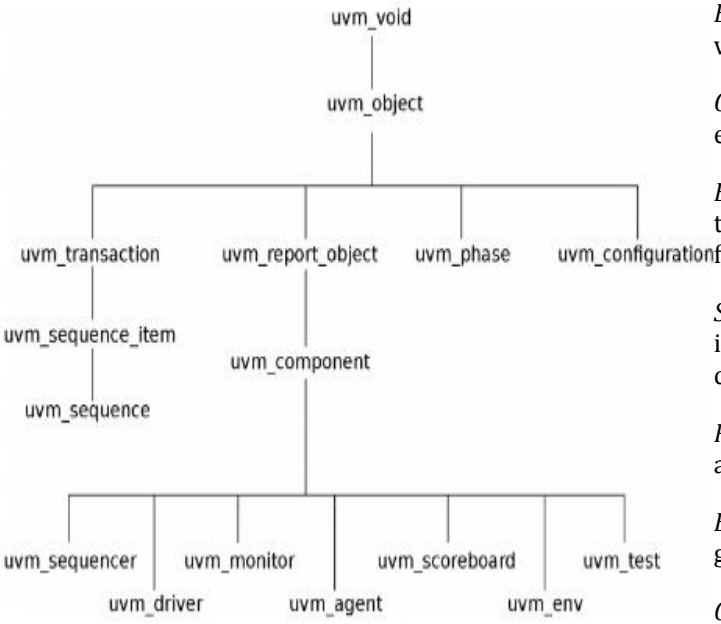

Figure 5.Partial UVM class tree [10]

3.3. UVM phases

There are different phases in UVM such as build phase, connect phase, end of elaboration phase, start of simulation phase, run phase, extract phase, check phase, report phase as shown in figure 6. All these phases are executed in an orderly manner. When a new class is derived, test bench simulation go through these different phases in order to build, configure and connect the components

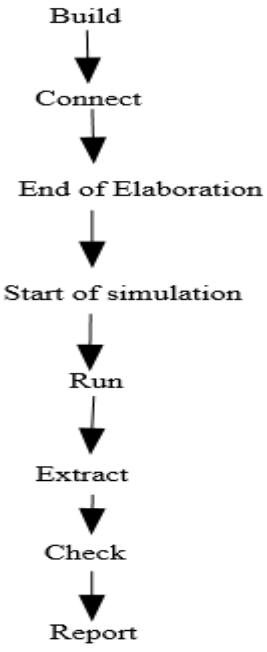

Figure 6.Partial list of UVM phases [10]

Build phase: It is used for construction and configuration of various components or ports or exports.

Connect phase: It is used for connecting various ports or exports of components.

End of elaboration phase: It is used to provide fine-tuning in the test benches, to print the topology and for opening the files.

Start of simulation phase: It gives notification to DUT to indicate that verification environment is completely configured and is ready.

Run phase: It is the main phase for execution. In run phase actual code is executed.

Extract phase: It is the phase where all information are gathered.

Check phase: It is the phase where extracted information results are checked.

Report phase: It is the phase where pass/fail status are checked.

4. RESULTS AND DISCUSSIONS

Verification plays very important role in VLSI. It is achieved by building test benches. The test benches are built in UVM. Simulation is carried out in Cadence Incisive Simulator tool. Here AXI4 slave is DUT written in Verilog and AXI4 master is used as test bench.

4.1. Simulation result of write operation

Master drives the address, and the slave accepts it one cycle later. The write starting address values are passed to slave are 00001000, 00011000, 00021000, 00031000 and 00041000 with AWID 001,002,003,004,005 as shown in figure 7. The data are sent as a burst from the master to slave. The slave calculates the subsequent transfer address based on the burst length, size and type as shown in figure 8. Five sequences of data are sent such as 00000001 to 00000004, 00010001 to 00010004, 00020001 to 00020004, 00030001 to 00030004 and 00040001 to 00040004.Each sequence has burst length of 4 (AWLEN + 1) and burst size of 4 bytes. The first sequence of write operation is as shown in figure 9.As per UVM Irun log file report, addresses are incremented by 4 according to burst length and burst size specified. The BID value is matching with the AWID value of the write transaction which indicates that slave is responding correctly. BRESP signal that is write response signal from slave is 0 which indicates OKAY.

4.2. Simulation result of read operation

The read starting address values are passed to slave are 00001000, 00011000, 00021000, 00031000 and 00041000 with AWID 001,002,003,004,005 as shown in figure 7. The data are sent as a burst from the slave to master. The slave calculates the subsequent transfer address based on the burst length, size and type as shown in figure 10. Five sequences of data are sent such as 00000001 to 00000004, 00010001 to 00010004, 00020001 to 000020004, 00030001 to 00030004 and 00040001 to 00040004. Each sequence has burst length of 4 (ARLEN + 1) and burst size of 4 bytes. The first sequence of read operation is as shown in figure 11.As per UVM Irun log file report, addresses are incremented by 4 according to burst length and burst size specified. The RID value is matching with the ARID value of the read transaction which indicates the slave is responding correctly. RLAST signal from slave indicates the last transfer in a read burst.

Figure 7. Simulation result of read/write address

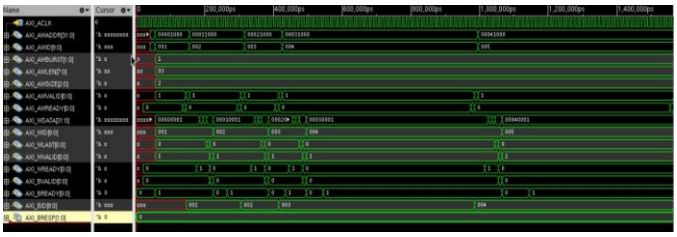

Figure 8. Simulation result of write transaction

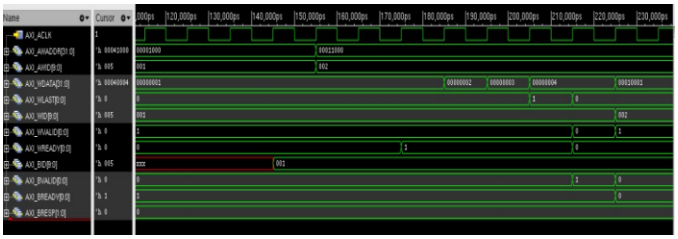

Figure9.simulation result of first sequence of write transaction

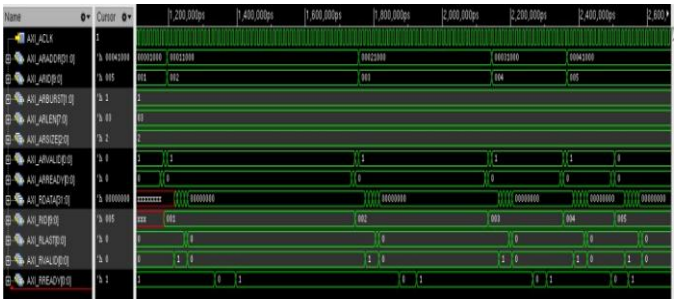

Figure10 simulation result of read transaction

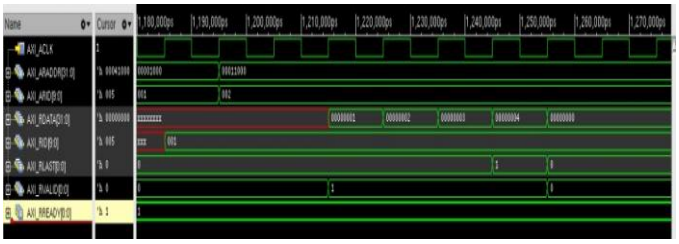

Figure 11.simulation result of first sequence of read transaction

4.3. UVM Irun log file report

```
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   88
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[AXI_master_driver]    22
[AXI_master_monitor]     6
[AXI_slave_monitor]    22
[CFGNRD]     1
[DEMO_AXI_master_write_data_after_write_addr]     1
[Demo_scoreboard]    20
[NOEXPECT]     4
[PHASESEQ]    13
[RNTST]     1
[TEST_DONE]     2
Simulation complete via $finish(1) at time 3125 NS + 45
/cad/INCISIV131/tools/uvm/uvm_lib/uvm_sv/sv/base/uvm_root.svh:457
$finish;
```

4.4.UVM report summary

```
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   88
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[AXI_master_driver]    22
[AXI_master_monitor]     6
[AXI_slave_monitor]    22
[CFGNRD]     1
[DEMO_AXI_master_write_data_after_write_addr]     1
[Demo_scoreboard]    20
[NOEXPECT]     4
[PHASESEQ]    13
[RNTST]     1
[TEST_DONE]     2
Simulation complete via $finish(1) at time 3125 NS + 45
/cad/INCISIV131/tools/uvm/uvm_lib/uvm_sv/sv/base/uvm_root.svh:457
$finish;
```

5. CONCLUSION

The AMBA AXI4 specification signifies a major evolutionary step in interconnect technology for System on-chip designs. The read and write operation of AXI4 slave has been verified for Single Master Slave implementing INCR burst type using Universal Verification methodology. Simulation results are shown in cadence Incisive Enterprise Simulator. The data is to be read or written to the AXI4 slave is assumed to be given by the AXI4 master and is read or written to a particular address location of AXI4 slave. UVM provides a rich set of base class library and features required for efficient verification. It gives an environment that is robust, easy to understand and thus, reusable by others vendors. Using UVM, requires less time to generate a testbench as it offers higher level of abstraction. It covers almost all the possible scenarios and corner cases and thus, increases the functional coverage.

Acknowledgment

The author express sincere gratitude to Mrs. Ashwini S Assistant Professor Dept. of ECE for continuous guidance and support. The author is also thankful to Department of Electronics & communication Engineering, NCET. The mechanism is implemented using cadence tool which is made obtainable by department.

REFERENCES

[1]Samir Palnitkar, Verilog HDL: A Guide to Digital Design and synthesis, 2nd Ed, Hall PTR Pub, 2003.

[2] ARM, AMBA Specifications (Rev2.0). [Online].Available at http://www.arm.com,1999

[3] ARM, AMBA AXI Protocol Specification (Rev 2.0). [Online]. Available at http://www.arm.com, March 2010

[4] Chien-Hung Chen, Jiun-Cheng Ju, and Ing-Jer Huang"A Synthesizable AXI Protocol Checker for SOC Integration,"IEEE, ISOCC 2010.

[5] Accellera Organization, "Universal Verification Methodology (UVM) 1.1 Class Reference", June 2011

[6]Manjula, R.B. ; Manvi, S.S. ; Kaunds, P. "Data transactions on system-on-chip bus using AXI4 protocol" Recent Advancements in Electrical, Electronics and Control Engineering (ICONRAEeCE), 2011 International Conference, 15-17 Dec. 2011

[7]V.N.M.Brahmanandam K, Choragudi Monohar, "Design of Burst Based Transactions in AMBA-AXI Protocol for SoC Integration," International Journal of Scientific & Engineering Integration International Journal of Scientific & Engineering Research Volume 3, Issue 7, July-2012

[8]Ms.Anusha Ranga, Mr. L. Hari Venkatesh,Mr.Venkanna, "Design and Implementation of AMBA-AXI Protocol using VHDL for SoC Integration," in International Journal of Engineering Research and Applications, Vol. 2,Issue4, July-August 2012, pp.1102-1106.

[9] T. Ananth Kumar, DR.S. Saraswathi Janaki, "Design of AXI Bus For 32-bit Processor using Bluespec", ISSN: 2278 – 1323 International Journal of Advanced Research in Computer

[10] Pedro Araujo. "Development of a Reconfigurable Multi-Protocol Verification Environment Using UVM Methodology",