DEVELOPMENT OF VIP FOR ADVANCED EXTENSIBLE INTERFACE (AXI 4) IN OPEN-POWER PROCESSOR BASED FABLESS SoC

Paseddula Kama Raju
M Tech Student
Department of ECE
JNTUACEA, Anantapur
Andhra Pradesh, India

B.C. Vengamuni M Tech
Assistant Professor (Adhoc)
Department of ECE
JNTUACEA, Anantapur
Andhra Pradesh, India.

ABSTRACT

Objective: Because of the recent trend of IP-based designs in System on Chip (SoC) development, testing the System on Chip is required (SoC). The Verification Intellectual Property (VIP) is particularly useful in this circumstance. Many SoC projects are adopting this trend of IP Core-based design flow and VIP-based verification flow to reduce time to market and speed up the SoC verification process. The primary goal of this effort is to create a Verification Intellectual Property (VIP) for the Advanced extensible Interface (AXI)-4.0 Protocol, a member of the Advanced Micro controller Bus Architecture family (AMBA).
Method: The most recent methodology, known as the Universal Verification Methodology UVM, is used to construct the VIP. Findings. All five of AXI's channels are successfully tested for functionality. Multiple Burst based, out of order, outstanding transaction completion scenarios are also verified with the help of Questa sim tool.

Keywords
AXI, Functional Coverage, Out of Order Transaction, SoC, VIP, AMBA, IP, Multiple Outstanding Transaction,

1. INTRODUCTION

In current years, because of the miniaturization of semiconductor procedure era and computing to continue to exist in cutting-edge marketplace conditions, regular customization is required. Semiconductor procedure era has been evolving at a quicker price given that 1971. Semiconductor procedure era became 10m, in 2010 the era fell to 32nm, and the destiny is shiny with 7nm. The Advanced Microcontroller Bus Architecture (AMBA) specification defines an on-chip communique modern for the layout of excessive widely wide-spread normal overall performance 32-bit and 16-bit embedded microcontrollers. It has turn out to be delivered thru way of manner of ARM Ltd in 1996 and is appreciably used as bus-to-chip verbal exchange in System on chip (SoC) designs. AMBA is a registered trademark of ARM Ltd. The first AMBA buses had been Advanced System Bus (ASB) and Advanced Peripheral Bus (APB). In its second release, AMBA-2, ARM delivered AMBA High widely wide-spread normal overall performance Bus (AHB) it's miles a single top clock protocol. In 2003, ARM delivered the third era, AMBA 3, which incorporates the Advanced Extensible Interface (AXI 3) for issue-to-issue connectivity. Later in 2010, ARM delivered the fourth era of AMBA 4, consists of a complex model Advanced extensible Interface (AXI 4) benefit even higher interconnections. The goal of the verification is to reveal the purposeful definition of a project. In the lifestyles of a business integrated circuit, validation consumes approximately 70% of the signing effort. The significance of verification may be calculated in phrases of the charges of layout mistakes which may be high. The current plus, the whole lot else spent withinside the schematic of the brand-new frames and included circuits is presently spent on validation with the excellent of the multifaceted body and exponentially increasing port numbers, the architects are dealing with the maximum complicated take a look at withinside the subject matter shape for approval. New improvements and take a look at plans inclusive of bodily blending and configuration reuse growing ever wider limits simplify the problem. Specialists are pressured to apply the great to be had approval and configuration gadgets to shorten the period of the verification process. The first manner to brief and secure body manage entails each the tool and the development of the procedure, however it is a lethal blend to anticipate that having basically the great gear will convey circuits included and excellent frames that meet marketplace needs and growth the goal weight.

2.VIP

As reusable, pre-designed logic blocks of IP-based designs are used in complicated designs, so too must the verification. A reusable verification model that can be introduced into the test bench and used to validate a design is known as a Verification Intellectual Property (VIP). Using a VIP will speed up the verification process, cut down on how long it takes to run the initial test case, and shorten the time it takes to launch the product. A pre-defined functional block known as a VIP can be characterized as a configurable component that can be modified by the user and is simple to integrate into various verification environments. VIP's will have the necessary elements for the generation of test bench and protocol checking mechanism. In proposed method, verification of all the memory transactions in AXI protocol for all the five channels i.e., read address and read data write address, write data, write response channels are done. In this work, the total test bench environment was modeled by using System Verilog and development of verification to the slave block in AXI is done by using UVM.

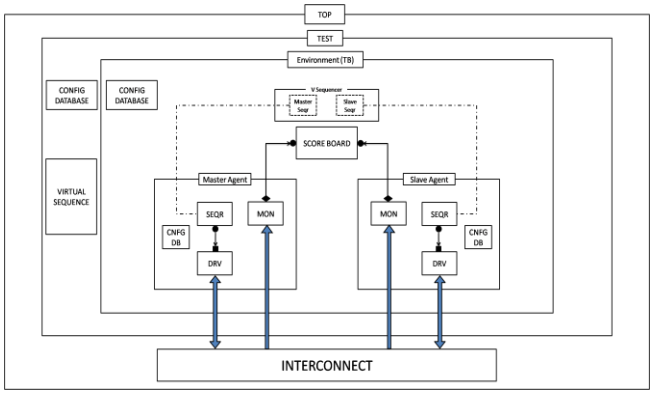


Fig: Proposed AXI VIP

3. Universal Verification Methodology

3.1 The Verification Components in UVM

Base classes and infrastructure tools make up the UVM library. The base classes present in the UVM hierarchy can be broadly divided into two groups: data and the component class hierarchy, which is descended from UVM components, is meant to simulate the testbench's permanent structures, such as monitors and drivers. The purpose of the data classes created from UVM sequence item is to model transactions and stimuli.

Design Under Test

The specifications of the design are what need to be verified. In terms of the designing language, this is basically the RTL description. The design's attributes and purposes are explained.

Sequencer

The entity on which the sequences will run is a sequencer. Sequence of transactions must be used in order to assess DUT behavior. When the driver requests it, the sequencer executes the stimulus creation code and passes the sequence items down to the driver.

Driver

The DUT signals are driven by the driver. It receives the sequencer's sequence elements and displays them on the interface. It is the environment's active component.

Monitor

The monitoring device is the verification environment's passive component. It gathers data as a packet, scans the incoming DUT signals on the interface, and then sends the packet to the coverage collector and scoreboard for coverage data.

Agent

Agent is an abstract container with a monitor, a sequencer, and a driver. It operates in two different ways: passively and actively. In active mode, it drives the signal to the DUT, and in passive mode, it scans the signals coming from the DUT.

Scoreboard

It is a verification component that compares the DUT response to the expected response in order to verify that it matches the expected response. It provides information on the proportion of times the response was accurate and incorrect.

Environment

The building is put together by it. Depending on the needs of the design, it incorporates one or more agents, a scoreboard, and other measurement and checking components.

Test

It has the highest position in the component hierarchy. UVM tests use the classes that are derived from a test class. The test class makes it possible to determine the dynamic behavior of the processes using sequences during the creation of the testbench and verification components.

Sequence items

These are the essential data elements that are transferred between the verification components at an abstract level.

Sequences

To create a true set of inputs, these are gathered from objects in the sequence. The transactions produced by sequences can be randomly generated or predetermined.

3.2 The Class Library Hierarchy in UVM

The environment has been fully configured and is ready to start run phase There are various run stages, which are utilized to execute simulations. As this phase takes more time, it is the only one that employs tasks to specify. All UVM sequences and components derive from the base class UVM object. All UVM components extend the UVM component class, which is derived from this class. The UVM transaction class extends sequence item and sequence, which are subclasses of the transaction class, which is derived from the UVM object class.

3.2.1 UVM Phases:

All of the components in the verification environment will be called in the following order during the predefine phases of UVM simulation.

Build phase: By instantiating the necessary components, it creates the environment's fundamental structure.

Connect phase: It is used in the child component to connect ports to exports, exports to ports, and ports to ports.

End of elaboration phase: It means the verification environment has been meticulously created and tweaked.

Start of simulation phase: It implies that verification was done on the DUT.

Extract phase: It is used to gather information from various locations inside the verification environment. It retrieves and extracts all of the data from the scoreboard.

Check phase: Any unexpected condition in a verification environment is verified.

Report phase: It provides a report of the specific test that was carried out.

Final phase: It provides information on the phases' completion and the possibility of ending the simulation.

4. Proposed Work

The environment (ENV) in System Verilog is not reusable for many test scenarios. Because different test scenarios necessitate different types of transactions. As a result, packet generation must be done independently of the environment (ENV). System Verilog does not support reusability or portability. In System Verilog, changing objects at the top level is not possible. There is no configuration facility for test bench infrastructure in System Verilog. For standard communication, System Verilog employs mail boxes. However, only the System Verilog language can recognize mailboxes. No other language understands mailbox. These are the disadvantages of System Verilog. All of these disadvantages can be overcome by employing the most modern approach, referred to as the Universal Verification Methodology (UVM). Methodology is a collection of best practices developed by verification experts through which a standard framework for the verification environment can be built. UVM has a collection of base class libraries. (For example, uvm_component, uvm_object, and so on.) To achieve interoperability, UVM has a standard communication mechanism. (For example, TLM ports.) The testbench in UVM can be configured from the top level. (For example, factory, configuration methods, and so on.) Configuration methods aid in the creation of a reusable testbench. UVM enables us to generate scenarios that are independent of the testbench environment. (For example, sequence, sequence_items, and so on.) We can achieve reusability in plug and play, which is used for on-chip communication, using UVM. AXI specifies higher performance as well as existing bus architectures. AXI is the most popular bus architecture in today's complex SoCs and FPGAs.

AXI protocol is appropriate for the following applications:

- For applications that require high frequency operation but do not necessitate the use of complicated bridges.
- for applications that have a wide range of components whose interface requirements they must satisfy for memory devices with a long initial access time.
- The AXI is used as a bus to interconnect the functional blocks inside a SoC
- Backward compatibility with existing APB and AHB components.
- To connect the functional blocks inside a SoC, the AXI is employed as a bus.

The special features of AXI-4 are as follows: -

- It uses five channels to spread its address, control and data phases separately.
- It supports sending unaligned data transfers via byte strobes.
- By issuing only the starting address, it employs burst-based transactions.
- It allows for the creation of multiple outstanding addresses.
- The completion of out-of-order transactions is supported.
- It has separate data channels for reading and writing.

Fig: Read and Write channels architecture

One that starts transfers and is capable of making decisions is referred to be a master device. A device that cannot make decisions and is obligated to carry out the master's orders is a slave. Five channels make up the AXI protocol, and they all function separately. Since the channels are independent of one another, there is no reliance. To enable quick data transport, the AXI protocol was created. These are the explanations behind it. i.e.,

(a) The write data channel can be used for a write transaction, and the read data channel can be used for a read transaction simultaneously. This enables separate and concurrent execution of write and read operations. AXI offers quicker data transfer speeds as a result, and this is one of the reasons behind this.

(b) The issue of numerous outstanding transactions is supported by the AXI-4 protocol. In other words, the master is allowed to start a new transaction before the previous one is finished. In this way, the master can start 'N' transactions, and each of these transactions that the master starts can be saved in a pipeline. As a result, the bus is more effective and data transfer speed increases. In light of this, memory systems with high initial access latencies are the ones for which AXI is most appropriate. By doing this, a slave with a slow initial access latency is prevented from obstructing the channel. This will enhance the bus's functionality,

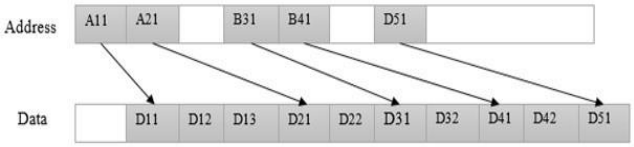


Fig: Several outstanding addresses Example of a transaction

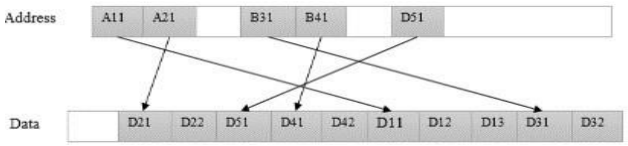


Fig: completion of Out of Order Transactions case.

(c) The AXI-4 protocol will also support Out_Of_Order(000) transaction completion in the read data and write response channels only. Data is always sent in the write data channel of AXI-4 in the order of the addressed slaves. Consider the first slave in the read data channel as being busy and unable to send data right away. The second slave, however, is prepared to send data. The second slave does not have to wait for the first slave to complete its duty as a result. The second slave can transfer data ahead of the first slave as a result. The first slave can transfer data once it is prepared. Therefore, speedier slaves can reply ahead of slower slaves. Transaction latency is drastically reduced.

Similar to this, quicker slaves can respond to the master via the write response channel first, completing that particular transaction. Any format of reply is capable of being sent by Slave. Depending on the situation, the slave might or might not answer in the same sequence as the master did. The bus's efficiency and speed are both enhanced by this. The Interconnect will make sure that the response is returned to the appropriate Master, whichever Master initiated the transaction by matching with the AWID displayed in the Figure waveforms.

In AXI, the Master device simply gives the initial address of the burst transaction; it is up to the slave to determine the succeeding addresses depending on the burst length and size of each transfer in the burst, which is started by the Master. The master will also assign an AWID to the address along with it. These AWIDs are used to identify the slaves that receive data and to return responses to the correct master. The VALID signal is turned HIGH when the master gives a legitimate address. The slave recognizes the master by changing the READY signal to HIGH when it is ready to take the signals delivered by it. Until the slave accepts the address as well as other control information by asserting the READY signal to HIGH, the master must hold onto signals like AWSIZE, AWBURST, AWLEN, and so forth. One channel contains multiple signals together. The master will send control signals such as AWBURST, AWSIZE, AWLEN, and others across the write address channel in addition to the burst's starting write address. The write data channel will be used by the master to communicate write data to the slave. Through the write response channel, the slave will acknowledge the master. The write response channel will be used by the slave to acknowledge the master. For the duration of the burst operation, only one response is transmitted to the master, and it is also sent via a different channel. In AXI, a transaction is deemed to be finished after the slave has responded to the master with a response regarding the transmitted data. Through the read address channel, the master will communicate the address and a few control signals, such as ARSIZE, ARBURST, ARLEN, and so on, to the slave. Based on the control information from the master, the slave will determine the subsequent addresses. The read data channel will now be used by Slave to transmit read data to Master. The slave must recognize each piece of read data. As a result, the slave can send read data and acknowledgment through the read data channel, which is a single channel. A read response channel is not necessary for AXI as a result. To make the design generator in the slave easier to use, the address is allotted to each slave in 4KB increments. Bursts are only allowed to be 4KB in size. Early burst termination is not the WLAST signal, whereas the last transfer in a read burst is shown by the RLAST signal. The WSTRB signal identifies which byte lanes in a write burst have valid data. For every eight bits of write data, there is one write strobe bit. The maximum amount of data bytes that can be transferred in each beat of transfer inside a burst is indicated by the burst size. An incrementing burst's initial transfer can be out of alignment, but all future transfers in the burst must be. For the burst's wrapping type, every transfer in the burst needs to be aligned. The fixed type of burst requires that each transfer use the same byte lane and that the address remain consistent during the burst. Aligned and unaligned transfers may also be issued using AXI. Only 2,4,8,16 bursts should be used for wrapping bursts. When the data size is less than 32 bits, a narrow transfer occurs.

The VIP was developed in this work without any Design Under Test (DUT). So, this VIP has one Slave agent and one Master agent, and both are ACTIVE agents, that means they have all of the monitor, driver, and sequencer. Each and every transaction initiated by the Master agent is assumed to be going to the other slave. The number of configured slaves can be from the test case by overwriting the required number of slaves value into the configuration class's "Number_of _slaves" parameter. In this case, the master agent behaves same as the master DUT, and the slave agent behaves same as the slave DUT. After the VIP is ready, it will operate according to the AXI-4 protocol. If a Master component (Master DUT) is provided, it must have a slave agent to communicate. Set the slave agent to only be ACTIVE while the master agent is set to PASSIVE in the VIP. It is only necessary to monitor the driven signals because the master DUT will drive the data. If a slave component is provided, make the slave agent in VIP a PASSIVE agent while keeping the master agent only as ACTIVE. If the master and slave both components are provided, we ensure to make both the slave and master agents as PASSIVE. We need to monitor these signals and check in the scoreboard.

5. Assertion Based Verification (ABV)

Assertions are used to validate the behavior of a design. Assertions document knowledge about how a design should work. Assertions are statements about properties that should be true. They also provide information about a design's functional coverage. Assertions are primarily used for protocol and functional testing. Assertions improve a design's observability and controllability. Assertions help with debugging. If there are no assertions and we find some kind of wrong behavior at design output, it is difficult to debug and figure out where the problem is However, if assertions are embedded to capture all of the design rules, that failure can be debugged much faster because the bug is caught right at the line of code. Assertions will detect bugs at the point where they occur. As a result, the verification process can be sped up with the help of assertions. As a result, the verification process can be completed in less time. The type of stimulus used influences assertions. This VIP examines assertions for dependencies between channel handshake signals.

6. Test Cases

All five channels are tested and verified in this paper. Basic Read and Write scenarios for all possible burst size and burst types. UVM also successfully validates Out of Order transaction completion and outstanding addresses transaction cases.

7. Results

The Master will issue four different AWADDR, AWLEN, AWSIZE, and AWBURST commands. WDATA and WSTRB are driven in response to the inputs provided by Master. The WLAST signal is driven high for one clock cycle after driving the last data of each burst to indicate the end of the burst. In Figure 6, the master will provide four distinct AWADDR, AWLEN, AWSIZE, and AWBURST values. Whichever burst completes the transaction first will respond to Master first by BID. BID will match with AWID to indicate that the response was returned to the proper master. The read channel architecture is depicted in Figure. ARADDR, ARLEN, ARSIZE, and ARBURST will be given by the master. Slave to Master drives RDATA and response based on these inputs.

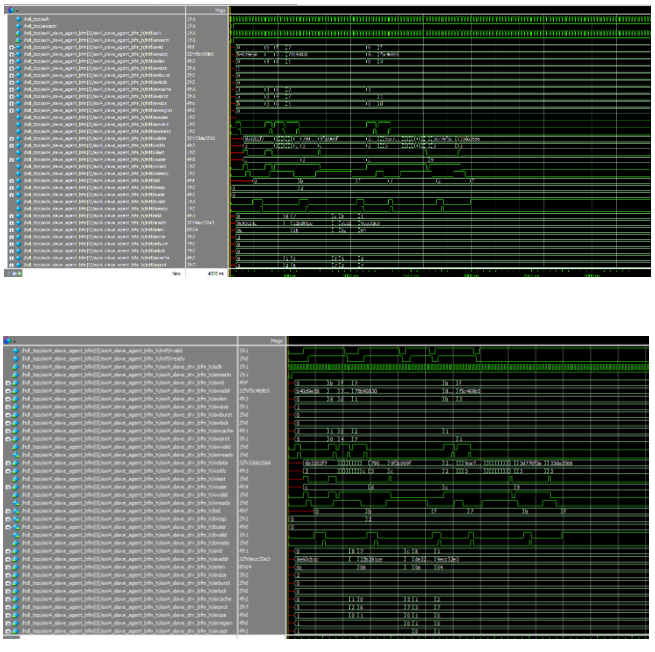
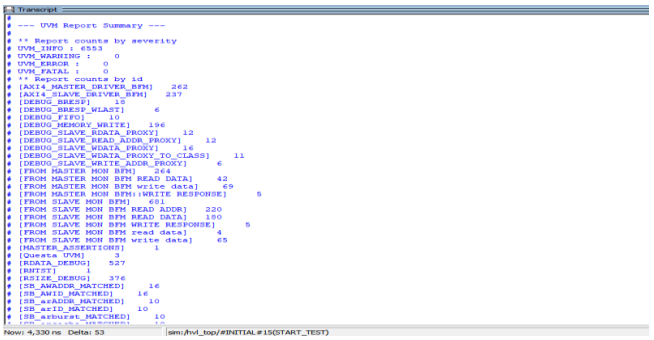
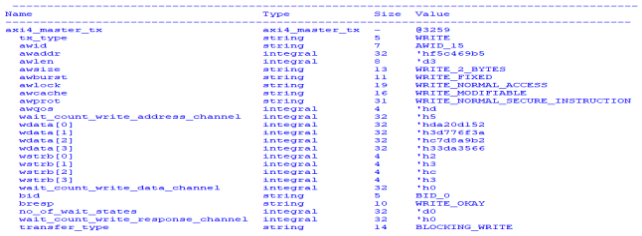


Figure . Waveforms which explain Read, write channel architecture.

UVM Report

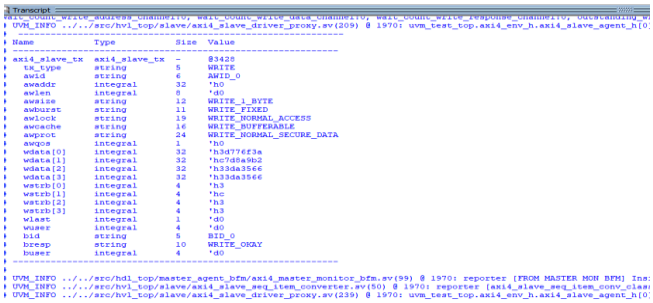


MASTER Transaction report



Name	Type	Size	Value
axi4_master_tx	axi4_master_tx	-	@3259
tx_type	string	5	WRITE
awid	string	7	AWID_15
awaddr	integral	32	'hf5c469b5
awlen	integral	8	'd3
awsize	string	13	WRITE_2_BYTES
awburst	string	11	WRITE_FIXED
awlock	string	19	WRITE_NORMAL_ACCESS
awcache	string	16	WRITE_MODIFIABLE
awprot	string	31	WRITE_NORMAL_SECURE_INSTRUCTION
awqos	integral	4	'hd
wait_count_write_address_channel	integral	32	'h5
wdata[0]	integral	32	'hda20d152
wdata[1]	integral	32	'h3d776f3a
wdata[2]	integral	32	'hc7d8a9b2
wdata[3]	integral	32	'h33da3566
wstrb[0]	integral	4	'h2
wstrb[1]	integral	4	'h3
wstrb[2]	integral	4	'hc
wstrb[3]	integral	4	'h3
wait_count_write_data_channel	integral	32	'h0
bid	string	5	BID_0
bresp	string	10	WRITE_OKAY
no_of_wait_states	integral	32	'd0
wait_count_write_response_channel	integral	32	'h0
transfer_type	string	14	BLOCKING_WRITE

SLAVE Transaction report



Name	Type	Size	Value
axi4_slave_tx	axi4_slave_tx	-	@3428
tx_type	string	5	WRITE
awid	string	6	AWID_0
awaddr	integral	32	'h0
awlen	integral	8	'd0
awsize	string	12	WRITE_1_BYTE
awburst	string	11	WRITE_FIXED
awlock	string	19	WRITE_NORMAL_ACCESS
awcache	string	16	WRITE_BUFFERABLE
awprot	string	24	WRITE_NORMAL_SECURE_DATA
awqos	integral	1	'h0
wdata[0]	integral	32	'h3d776f3a
wdata[1]	integral	32	'hc7d8a9b2
wdata[2]	integral	32	'h33da3566
wdata[3]	integral	32	'h33da3566
wstrb[0]	integral	4	'h3
wstrb[1]	integral	4	'hc
wstrb[2]	integral	4	'h3
wstrb[3]	integral	4	'h3
wlast	integral	1	'd0
wuser	integral	4	'd0
bid	string	5	BID_0
bresp	string	10	WRITE_OKAY
buser	integral	4	'd0

8. Conclusion & future scope

This work developed, simulated and verified of AXI-4 which utilized A2O processor based fabless SoC in SV and UVM using Questasim. Using AXI4, this suggested design could enhance data transfer efficiency. In Xilinx Vivado, the design is developed and interfaced with A20, while Mentor Questa is used for simulation.

9. References

1. Mahesh G, Sakthivel SM. Functional Verification of the Axi2Ocp Bridge using System Verilog and effective bus utilization calculation for AMBA AXI 3.0 Protocol. IEEE Sponsored 2nd International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS). 2015.
2. Naidu KJ, Srikanth M. Design and Verification of Slave block in Ethernet Management Interface using UVM. Indian Journal of Science and Technology. 2016 Feb; 9(5):1-7.
3. AMBA AXI-4 Specification, Copyright ARM Limited. http://www.gstitt.ece.ufl.edu/courses/eel4720_5721/labs/refs/AXI4_specification.pdf.
4. Sebastian R, Mary SR, Gayathri M, Thomas A. Assertion Based Verification of SGMII IP Core incorporating AXI Transaction Verification Model. International Conference

on Control, Communication and Computing India (ICCC). 2015; p. 585-88.

5. Chen X, Xia Z and Wang XA. Development of Verification Environment for AXI Bus Using System Verilog. International Journal of Electronics and Electrical Engineering. 2013; 2(1):112-14.
6. Chen CH, Iu JC, Huang II. A Synthesizable AXI Protocol Checker for SoC Integration. IEEE Trans, ISOCC. 2010; 8:103-6.
7. Ranga A, Venkatesh LH, Venkanna V. Design and Implementation of AMBA-AXI Protocol Using VHDL for SOC Integration. International Journal of Engineering Research and General Science. 2012; 2(4):1-5.