International Journal of Communication Networks and Security

Volume 1 | Issue 3 Article 8

January 2012

Design of AMBA AXI4 protocol for System-on-Chip communication

Shaila S Math
Dept. of ECE, REVA Institute of Technology and Management, Bangalore, India., shaila.s.math@gmail.com

Manjula R B
Dept. of ECE, REVA Institute of Technology and Management, Bangalore, India.,
manjularb@rediffmail.com

Recommended Citation

Math, Shaila S and B, Manjula R (2012) "Design of AMBA AXI4 protocol for System-on-Chip communication," *International Journal of Communication Networks and Security*: Vol. 1 : Iss. 3 , Article 8.
DOI: 10.47893/IJCNS.2012.1033
Available at: https://www.interscience.in/ijcns/vol1/iss3/8

Design of AMBA AXI4 protocol for System-on-Chip communication

Shaila S Math and Manjula R B
Dept. of ECE,
REVA Institute of Technology and Management, Bangalore, India.
shaila.s.math@gmail.com and manjularb@rediffmail.com

***Abstract*— Advanced microcontroller bus architecture (AMBA) protocol family provides metric-driven verification of protocol compliance, enabling comprehensive testing of interface intellectual property (IP) blocks and system-on-chip (SoC) designs. The AMBA advanced extensible interface 4 (AXI4) update to AMBA AXI3 includes the following: support for burst lengths up to 256 beats, updated write response requirements, removal of locked transactions and AXI4 also includes information on the interoperability of components. AMBA AXI4 protocol system supports 16 masters and 16 slaves interfacing. This paper presents a work aimed to design the AMBA AXI4 protocol modeled in Verilog hardware description language (HDL) and simulation results for read and write operation of data and address are shown in Verilog compiler simulator (VCS) tool. The operating frequency is set to 100MHz. Two test cases are run to perform multiple read and multiple write operations. To perform single read operation module takes 160ns and for single write operation it takes 565ns.**

Keywords- System-on-chip(SoC), Intellactual Property (IP), AMBA, AXI, VCS, Verilog.

I. INTRODUCTION

In recent years due to the miniaturization of semiconductor process technology and computation for survival in the current market conditions constant customization is required. The semiconductor process technology is changing at a faster pace during 1971 semiconductor process technology was 10µm, during 2010 the technology is reduced to 32nm and future is promising for a process technology with 10nm. Intel, Toshiba and Samsung have reported that the process technology would be further reduced to 10nm in the future. So with decreasing process technology and increasing consumer design constraints SoC has evolved, where all the functional units of a system are modelled on a single chip.

SoC buses [1] are used to interconnect an Intellectual Property (IP) core to the surrounding interface. These are not real buses, but they reside in Field Programmable Gate Array (FPGA). The AMBA [2] data bus width can be 32, 64, 128 or 256 byte, address bus width will be 32bits wide. The AMBA AXI4 [3] specification to interconnect different modules in a SoC was released in March 2010.

A. AMBA AXI4 architecture

AMBA AXI4 [3] 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. AMBA AXI4 system consists of master, slave and bus (arbiters and decoders). The system consists of five channels namely write address channel, write data channel, read data channel, read address channel, and write response channel. The AXI4 protocol supports the following mechanisms:

- Unaligned data transfers and up-dated write response requirements.
- Variable-length bursts, from 1 to 16 data transfers per burst.
- A burst with a transfer size of 8, 16, 32, 64, 128, 256, 512 or 1024 bits wide is supported.
- Updated AWCACHE and ARCACHE signalling details.

Each transaction is burst-based which has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. Table 1[3] gives the information of signals used in the complete design of the protocol.

The write operation process starts when the master sends an address and control information on the write address channel as shown in fig. 1. The master then sends each item of write data over the write data channel. The master keeps the VALID signal low until the write data is available. 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 signal BRESP[1:0] back to the master to indicate that the write transaction is complete. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.

After the read address appears on the address bus, the data transfer occurs on the read data channel as shown in fig. 2. 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. The RRESP[1:0] signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.

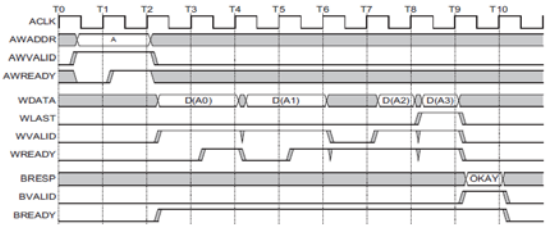

Figure 1: Write address and data burst.

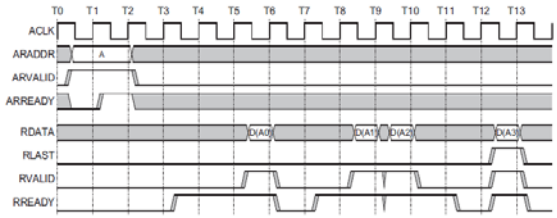

Figure 2: Read address and data burst.

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.

The protocol supports 16 outstanding transactions, so each read and write transactions have ARID[3:0] and AWID [3:0] tags respectively. Once the read and write operation gets completed the module produces a RID[3:0] and BID[3:0] tags. If both the ID tags match, it indicates that the module has responded to right operation of ID tags. ID tags are needed for any operation because for each transaction concatenated input values are passed to module.

B. Comparison of AMBA AXI3 and AXI4

AMBA AXI3 protocol has separate address/control and data phases, but AXI4 has updated write response requirements and updated AWCACHE and ARCACHE signaling details. AMBA AXI4 protocol supports for burst lengths up to 256 beats and Quality of Service (QoS) signaling. AXI4 has additional information on Ordering requirements and details of optional user signaling. AXI3 has the ability to issue multiple outstanding addresses and out-of-order transaction completion, but AXI4 has the ability of removal of locked transactions and write interleaving. One major up-dation seen in AXI4 is that, it includes information on the use of default signaling and discusses the interoperability of components which can't be seen in AXI3.

In this paper features of AMBA AXI4 listed above are designed and verified. The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 of this paper, discusses proposed work. In Section 4, simulation parameters are discussed. Section 5 discusses results. Future scope and concluding remarks are given in Section 6.

II. RELATED WORK

In a SoC, it houses many components and electronic modules, to interconnect these a bus is necessary. There are many buses introduced in the due course some of them being AMBA [2] developed by ARM, CORE CONNECT [4] developed by IBM, WISHBONE [5] developed by Silicore Corporation, etc. Different buses have their own properties the designer selects the bus best suited for his application.

The AMBA bus was introduced by ARM Ltd in 1996 which is a registered trademark of ARM Ltd. Later advanced system bus (ASB) and advanced peripheral bus (APB) were released in 1995, AHB in 1999, and AXI in 2003[6].

AMBA bus finds application in wide area. AMBA AXI bus is used to reduce the precharge time using dynamic SDRAM access scheduler (DSAS) [7]. Here the memory controller is capable of predicting future operations thus throughput is improved.

Efficient Bus Interface (EBI) [8] is designed for mobile systems to reduce the required memory to be transferred to the IP, through AMBA3 AXI. The advantages of introducing Network-on-chip (NoC) within SoC such as quality of signal, dynamic routing, and communication links was discussed in [9].

To verify on-chip communication properties rule based synthesizable AMBA AXI protocol checker [10] is used.

III. PROPOSED WORK

The work carried out in this project is the achievement of communication between one master and one slave. AMBA AXI4 slave is designed with operating frequency of 100MHz, which gives each clock cycle of duration 10ns. To access slave interconnect is needed, hence interconnect signals are also studied. Master block functions are assumed to be available and the slave characteristics are studied. The AMBA AXI4 system components consists of

1) Master
2) AMBA AXI4 Interconnect
   2.1) Arbiters
   2.2) Decoders
3) Slave

The master is connected to the interconnect using a slave interface and the slave is connected to the interconnect using a master interface as shown in fig. 3. The AXI4 master gets connected to the AXI4 slave interface port of the interconnect and the AXI slave gets connected to the AXI4 Master interface port of the interconnect. The parallel capability of this interconnects enables master M1 to access one slave at the same as master M0 is accessing the other.

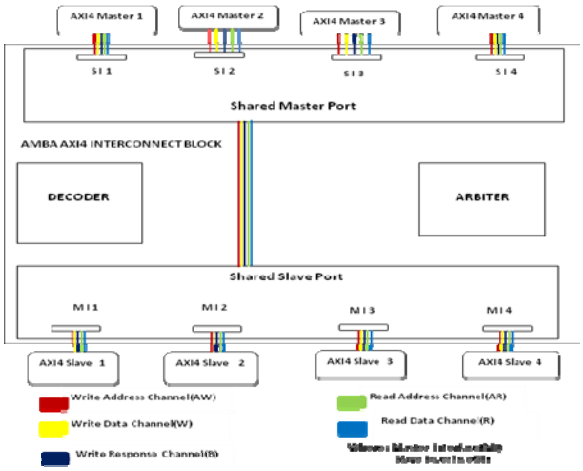

Figure 3: Block diagram of AMBA AXI4 bus interconnect.

A. *AMBA AXI4 master*

To perform write address and data operation the transaction is initiated with concatenated input of [awaddr, awid, awcache, awlock, awprot, awburst]. On the same lines for read address and data operations the concatenated input is [araddr, arid, arcache, arlock, arprot, arburst]. The addresses of read and write operations are validated by VALID signals and sent to interface unit.

B. AMBA AXI4 Interconnect

The interconnect block consists of arbiter and decoder. When two masters initiate a transaction simultaneously, the arbiter gives priority to access the bus. The decoder decodes the address sent by master and the control goes to one slave out of 16. The AMBA AXI interface decoder is centralized digital block. The decoder decodes the address sent by master and goes to one slave out of 16. 0-150 locations are meant for slave-1, next 151-300 addressable locations are meant for slave-2,… and so on till slave-16.

C. AMBA AXI4 slave read/write block diagram

The AXI4 slave consists of common read/ write buffer which stores the read/ write address and data as shown in fig. 4. Pending read address register stores the remaining read addresses to be sent; pending write address register which stores the remaining write addresses to be sent and pending write data register which stores the remaining write data to be sent. The read/write state machines receive internal inputs from the read/ write buffer. The AXI4 slave test bench initiates the read or write transaction and the output from the AXI4 slave are standard read/write channel signals. The AXI4 slave receives the write data in the same order as address.

Signals used to design slave module is shown in fig. 5. The test layer shown in the fig. 5 has 2 test cases. The test case 1:- for multiple read operations and case 2:- for multiple write operations.

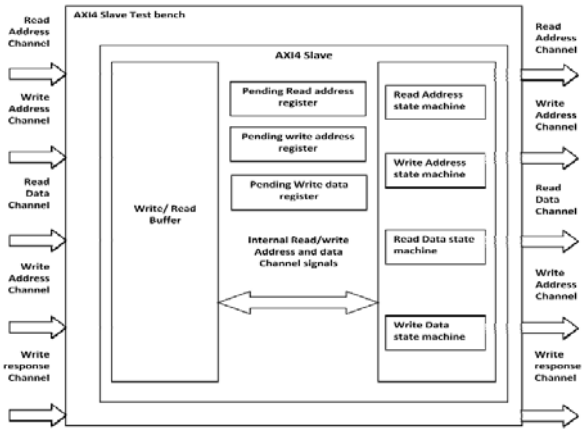

Figure 4: AMBA AXI4 slave Read/Write block Diagram.

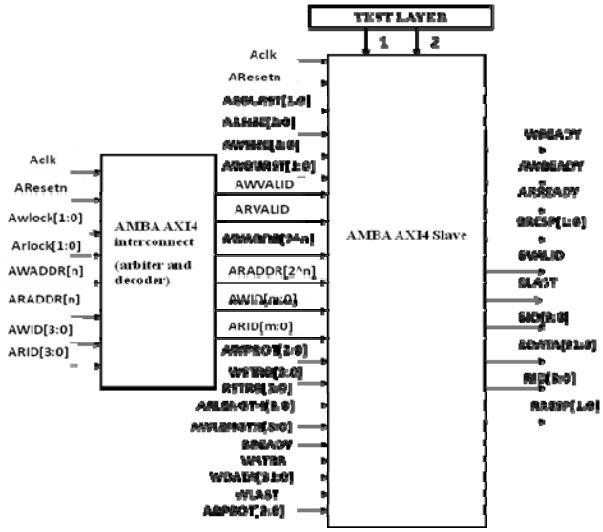

Figure 5: Signals used to design AMBA AXI4 interconnect and slave.

IV. SIMULATION

Simulation is being carried out on VCS tool [11] which is trademark of Synopsys, using Verilog [12] as programming language. The test case is run for multiple operations and the waveforms are visible in discovery visualization environment (DVE).

A. *Simulation inputs*

To perform multiple write and read operations, the concatenated input format and their values passed to invoke a function is shown in the fig. 6 and 7 respectively. Here the normal type of the burst is passed to module. Internal_lock value is 0, internal_burst value is 1 and internal_prot value is 1,for both read and write operations, which indicate that the burst is of normal type. For write operation address locations passed to module are 40, 12, 35, 42 and 102; for read operations 45, 12, 67 and 98.

```
Write operation transaction concatenated input format

module axi_slave(internal_awlck, internal_awid
internal_awcache, internal_awlock, internal_awprot, interna _awburst);

al.v1.execute_write_transaction(40,11,3,0,1,1);
al.v1.execute_write_transaction(12,4,3,0,1,1);
al.v1.execute_write_transaction(35,5,3,0,1,1);
al.v1.execute_write_transaction(42,6,3,0,1,1);
al.v1.execute_write_transaction(102,"_3,0,1,1);
```

Figure 6: Interface declaration and write function invocation.

```
Read operation transaction concatenated input format:

module axi_slave(internal_araddr, internal_arid,
internal_arcache,internal_arlock,internal_arprot, internal_arburst);

al.v1.execute_read_transaction(45,5,3,0,1,1 );
al.v1.execute_read_transaction(12,3,3,0,1,1);
al.v1.execute_read_transaction(67,8,3,0,1,1);
al.v1.execute_read_transaction(98,5,3,0,1,1);
```

Figure 7: Interface declaration and read function invocation.

B. *Simulation outputs*

The simulation output signals generated are as follows:

- From input side the validating signals AWVALID/ARVALID signals are generated by interconnect which gives the information about valid address and ID tags.
- For write operations BRESP[1:0] response signal generated from slave indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLERR, and DECERR.
- For read operations RLAST signal is raised by slave for every transaction which indicates the completion of operation.

V. RESULTS

Simulation is carried out in VCS tool and Verilog is used as programming language.

A. Simulation result for write operation

The AResetn signal is active low. Master drives the address, and the slave accepts it one cycle later.

The write address values passed to module are 40, 12, 35, 42 and 102 as shown in fig. 8 and the simulated result for single write data operation is shown in fig. 9. Input AWID[3:0] value is 11 for 40 address location, which is same as the BID[3:0] signal for 40 address location which is identification tag of the write response. The BID[3:0] value is matching with the AWID[3:0] value of the write transaction which indicates the slave is responding correctly. BRESP[1:0] signal that is write response signal from slave is 0 which indicates OKAY. Simulation result of slave for multiple write data operation is shown in fig. 10.

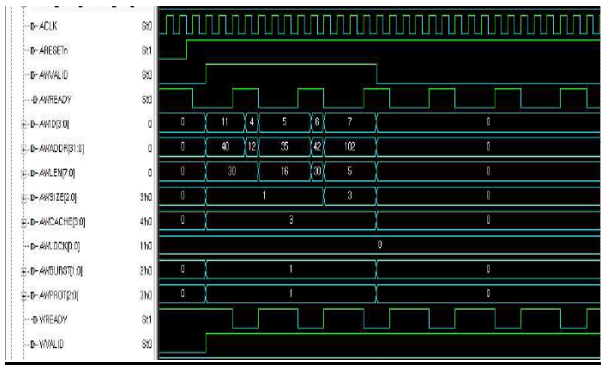

Figure 8: Simulation result of slave for write address operation.

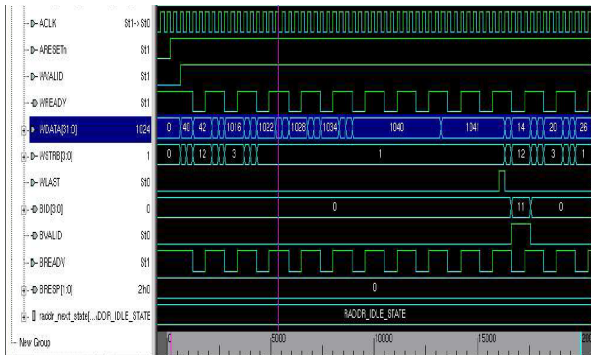

Figure 9: Simulation result of slave for single write data operation.

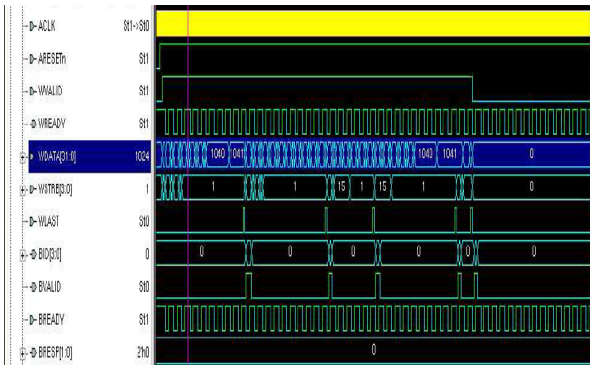

Figure 10: Simulation result of slave for multiple write data operation.

B. *Simulation result for read operation*

The read address values passed to module are 45, 12, 67, 98 as shown in fig. 11 and the simulated result for single read data operation is shown in fig. 12.

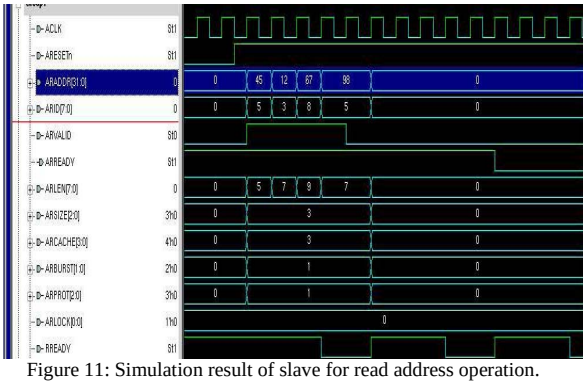

Figure 11: Simulation result of slave for read address operation.

Input ARID[3:0] value is 3 for 12 address location, which is same as the RID[3:0] signal for 12 address location which is identification tag of the write response. The RID[3:0] and ARID[3:0] values are matching, which indicates slave has responded properly. RLAST signal from slave indicates the last transfer in a read burst. Simulation result of slave for multiple read data operation is shown in fig. 13.

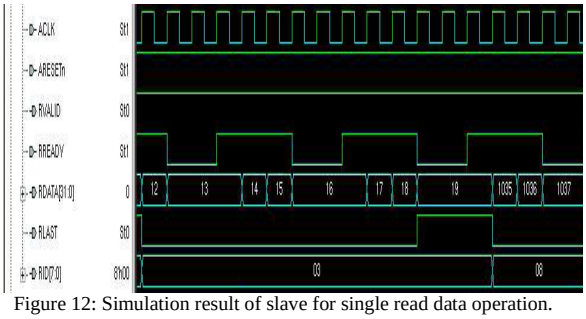

Figure 12: Simulation result of slave for single read data operation.

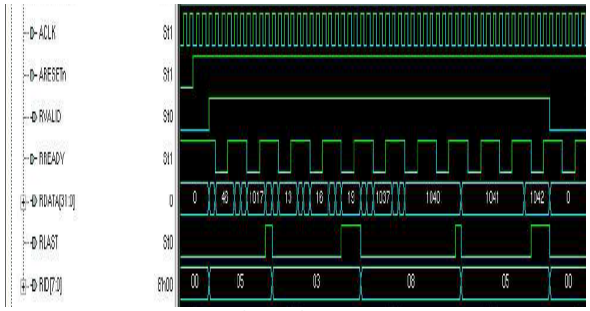

Figure 13: Simulation result of slave for multiple read data operation.

VI. Conclusion and Future scope

A. *Future scope*

The AMBA AXI4 has limitations with respect to the burst data and beats of information to be transferred. The burst must not cross the 4k boundary. Bursts longer than 16 beats are only supported for the INCR burst type. Both WRAP and FIXED burst types remain constrained to a maximum burst length of 16 beats. These are the drawbacks of AMBA AXI4 system which need to be overcome.

B. *Conclusion*

AMBA AXI4 is a plug and play IP protocol released by ARM, defines both bus specification and a technology independent methodology for designing, implementing and testing customized high-integration embedded interfaces. The data to be read or written to the slave is assumed to be given by the master and is read or written to a particular address location of slave through decoder. In this work, slave was modeled in Verilog with operating frequency of 100MHz and simulation results were shown in VCS tool. To perform single read operation it consumed 160ns and for single write operation 565ns.

References

[1] Shaila S Math, Manjula R B, “Survey of system on chip buses based on industry standards”, Conference on Evolutionary Trends in Information Technology(CETIT), Bekgaum,Karnataka, India, pp. 52, May 2011

[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] IBM, Core connect bus architecture. IBM Microelectronics. [Online]. Available: http://www.ibm.com/chips/products/coreconnect, 2000

[5] Silicore Corporation, Wishbone system-on-chip (soc) interconnection architecture for portable ip cores, (Rev B.3). [Online]. Available at http://www.opencores.org/projects.cgi/web/wishbone/wishbone, Sept 2002

[6] ARM, AMBA AXI protocol specifications, Available at, http://www.arm.com, 2003

[7] Jun Zheng, Kang Sun , Xuezeng Pan, and Lingdi Ping “Design of a Dynamic Memory Access Scheduler”, IEEE transl, Vol 7, pp. 20-23, 2007

[8] Na Ra Yang, Gilsang Yoon, Jeonghwan Lee, Intae Hwang, Cheol Hong Kim, Sung Woo Chung and Jong Myon Kim, “Improving the System-on-a-Chip Performance for Mobile Systems by Using Efficient Bus Interface”, IEEE transl, International Conference on Communications and Mobile Computing, Vol 4, pp. 606-608, March 2009

[9] Bruce Mathewson “The Evolution of SOC Interconnect and How NOC Fits Within It”, IEEE transl, DAC,2010, California, USA,Vol 6, pp. 312-313, June 2010

[10] Chien-Hung Chen, Jiun-Cheng Ju, and Ing-Jer Huang, “A Synthesizable AXI Protocol Checker for SoC Integration”, IEEE transl, ISOCC, Vol 8, pp.103-106, 2010

[11] Synopsys, VCS / VCSi User Guide Version 10.3[Online]. Available at, www.synopsys.com, 2005

[12] Samir Palnitkar, Verilog HDL: A Guide to Digital Design and synthesis, 2nd ed, Prentice Hall PTR Pub, 2003