DESIGN AND VERIFICATION ENVIRONMENT FOR AMBA AXI PROTOCOL FOR SOC INTEGRATION

Pradeep S R[1], Laxmi C[2]

[1]*M.Tech Student, Department of P.G Studies, VTU Gulbarga, Karnataka, India*
[2]*Guest Lecturer, Department of P.G Studies, VTU Gulbarga, Karnataka, India*

Abstract

Advanced microcontroller bus architecture (AMBA) protocol family provides a metric-driven verification of protocol compliance, enabling the comprehensive testing of interface intellectual property (IP) blocks and system-on-chip (SoC) design. The AMBA advanced extensible interface 4 (AXI4) update to AMBA AXI3 includes: the support for burst lengths up to 256 beats. It is updated write response requirements and removal of locked transactions. Verification has become the dominant cost in the design process. This paper proposes a work, how to build up the verification environment of AXI bus using SystemVerilog is introduced. Functional coverage, score-boarding and assertions is implemented with the proposed integrated verification environment.

Keywords: *AMBA, AXI, Verification, System Verilog etc…*

---***---

1. INTRODUCTION

There are many companies that develop core IP for SoC products. The interfaces to these cores can differ from company to company and may generally be proprietary in nature. The SoC developer then should expend time, effort, and cost to make “bridge” or “glue” logic that enables all of the cores within the SoC to communicate properly with each other. Incompatible interfaces are therefore barriers to each IP developers and SoC developers.

Integrated circuits have entered the era of System-on-a-Chip (SoC), which refers to integrating all components of a computer or other electronic system into a single chip. It contains digital, analog, mixed signal, and often radio-frequency functions – all on a single chip substrate. By increasing the design size, IP is an inevitable alternative for SoC design. And therefore the widespread use of all kinds of IPs has changed the nature of the design flow, making On-Chip Buses (OCB) essential to the design.

To speed up SoC integration and promote IP reusability, many bus-based communication architecture standards have emerged over the past several years. Since the first 1990s, many onchip bus-based communication architecture standards are projected to handle the communication needs of emerging SoC design. Some of the popular standards include ARM Microcontroller Bus Architecture (AMBA) versions of 2.0 and 3.0, IBM Core Connect, STMicroelectronics STBus, Sonics SMARRT Interconnect, Open Cores Wishbone, and Altera Avalon [2]-[6]. On the other side, the designers simply integrate their owned IPs with third party IPs into the SoC to significantly reduce design cycles. However, the main issue is that a way to efficiently ensure the IP functionality, that works properly after integrating to the corresponding bus architecture.

The AMBA AXI protocol is a standard bus protocol and most of the semiconductor companies’ design interconnects which supports AXI bus interface. AXI protocol is complex protocol because of its ultra-high-performance. On current projects, verification engineers are maximum number designers, with this ratio reaching 2 or 3 to one for the most complex designs. Therefore an efficient verification environment is needed [9].

Verification of such a complex protocol is challenging. This can be easily verified using the verification environment. This verification environment can be reused for other IPs also.

1.1 AMBA AXI4 Architecture

The AMBA AXI protocol is aimed towards high-frequency system designs and includes a number of features that make it suitable for a high - speed submicrons interconnect. In this project proposes a feature that supports a maximum of 256 data transfers per burst [3]. In AMBA AXI4 system 16 masters and 16 slaves are interfaced. Every master and slave has their own 4 bit ID tags. The system consists of master, slave and Interconnect bus [4]. The AXI4 protocol supports the following mechanisms:

•Two kinds of address mode: aligned and unaligned.
•Three types of burst: FIXED, INCR and WRAP.
•Sixteen choices of burst length in the range of 1-256.
•Four varieties of response types: OKAY, EXOKAY, SLVERR and DECERR.

Figure 1gives the information of signals used in the complete design of the protocol [3]. 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 information is transferred between master and slave using a write data channel to the slave or a read data channel to the master [8].

Signal	Source	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[1:0]	Master	Input	Write cache type
AWPROT[2:0]	Master	Input	Write protection
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
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
RVALID	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

Fig -1: Signal descriptions of AMBA AXI Protocol

2. RELATED WORK

The Advanced Microcontroller Bus Architecture (AMBA) is a protocol that is used as an open standard; on-chip interconnects specification for the connection and management of functional blocks in a system-on-chip (SoC). The AMBA bus is applied easily to small scale SoCs. Therefore, the AMBA bus has been the representative of the SOC market though the bus efficiency.

Three distinct buses are defined within the AMBA specification:

1. Advanced Peripheral Bus (APB).
2. Advanced High performance Bus (AHB).
3. Advanced extensible Interface Bus (AXI).

The AMBA specification defines all the signals, transfer modes, structural configuration, and other bus protocol details for the APB, AHB, and AXI buses.

The AMBA APB is used for interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface. APB peripherals can be integrated easily into any design flow, with the following specification:

- Peripheral bus for low-speed devices
- Synchronous, non multiplexed bus
- Single master (bridge)
- 8, 16, 32-bit data bus
- 32-bit address bus
- Non-pipelined

AMBA AHB is a new level of bus which sits above the APB and implements the features required for high performance, high clock frequency systems, with the following specification:

- Burst transfers
- Split transactions
- Single cycle bus master handover
- Single clock edge operation
- Wider data bus configurations (64/128 bits)

AXI extends the AHB bus with advanced features to support the next generation of high performance SoC designs. The goals of the AXI bus protocol include supporting high frequency operation without using complex bridges, flexibility in meeting the interface, and performance requirements of a diverse set of components, and backward compatibility with AMBA AHB and APB interfaces. The features of the AXI protocol are:

- Separate address/control and data phases
- Support for unaligned data transfers
- Ability to issue multiple outstanding addresses
- Out-of-order transaction completion.

3. PROPOSED WORK

The work is proposed in this project is the achievement of communication between one master and one slave using Verilog, then verifying the design using System Verilog.

3.1 Design of AXI Protocol

AMBA AXI4 slave is designed with operating frequency of 100MHz, which gives each clock cycle of duration 10ns and it supports a maximum of 256 data transfers per burst. The AMBA AXI4 system component consists of a master and a slave as shown in Figure 2.

There are 5 different channels between the AXI master and AXI slave namely write address channel, write data channel, read data channel, read address channel, and write response channel.

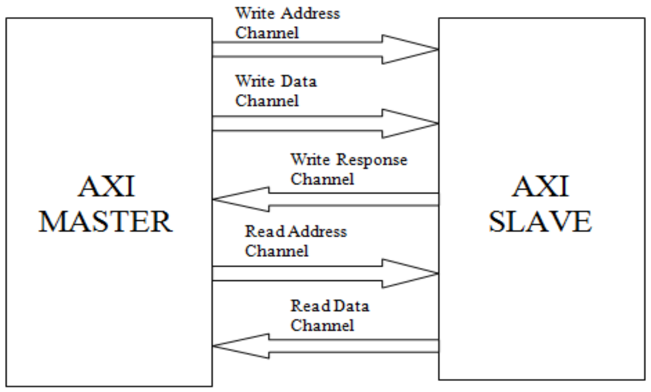


Fig -2: Block Diagram of a system

In AXI 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.

3.1.1 Address Write Channel (AW Channel)

AXI_MASTER drives the write command signals only when ARESETn is HIGH, else it drives all signals as zero. The address write command signals driven by the AXI_MASTER are - AWID,AWADDR, AWBURST, AWLEN, AWSIZE, AWCACHE, AWLOCK, AWPROT, with AWVALID as HIGH indicating that the driven signals are valid. The AXI_MASTER does not drive the AWVALID signal as LOW, until it receives the AWREADY signal, which is driven by the DESTINATION_SLAVE, indicating that, it has received the address write command signals. If AWREADY is LOW, then AXI_MASTER retains the same values. Figure 3 shows the state diagram for the address write command signals.

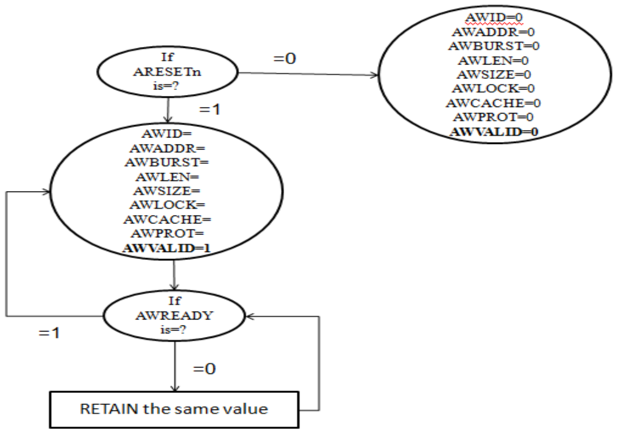


Fig -3: State diagram of Address Write Channel

3.1.2 Write Data Channel (W Channel)

The AXI MASTER drives these Write Data signals, after sending the write address command signals. It drives these signals, only when ARESETn is HIGH, otherwise it drives all signals to zero. AXI MASTER drives the WDATA signal with WVALID as HIGH, it holds the same value until it receives the WREADY signal. If WREADY is HIGH, it drives the next WDATA. AXI MASTER drives the AWLEN No. of data. While driving the last data it drives the WLAST as HIGH. Figure 4 shows the state diagram for the WRITE DATA channels.

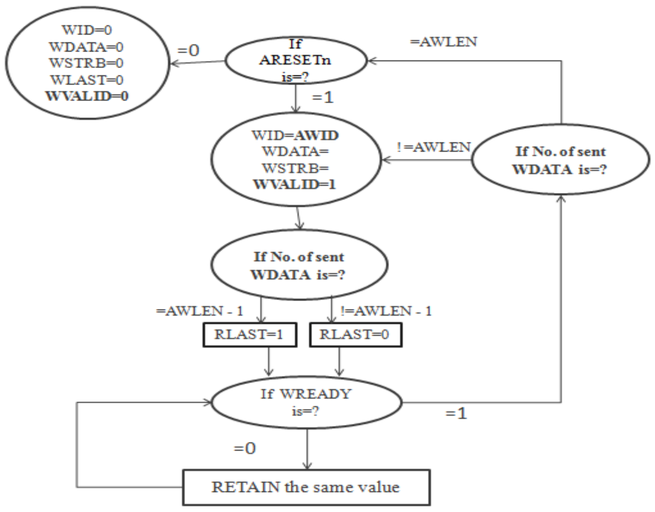


Fig -4: State diagram of Write Data Channel

3.1.3 Write Response Channel (B Channel)

The DESTINATION_SLAVE drives these Write Response signals, only when ARESETn is HIGH, otherwise it drives all signals as zero. DESTINATION_SLAVE waits for WLAST signal. After receiving the WLAST signal, it drives these response signals, with BVALID as HIGH. It holds the same value until it receives the BREADY signal from the AXI MASTER. If BREADY is HIGH, it drives all the signals as zero, at next positive edge of ACLK, otherwise it retains the same value. Figure 5 shows the state diagram for the Write Response channels.

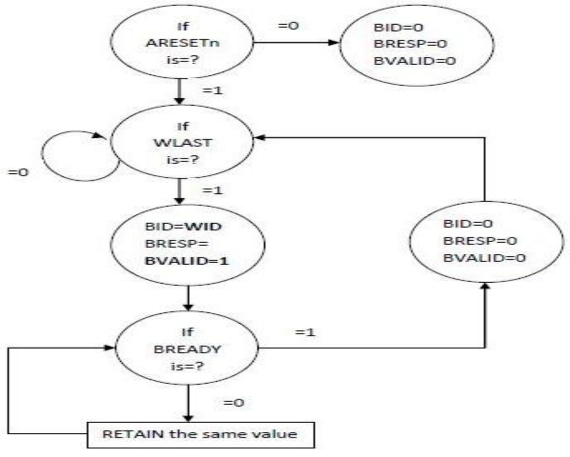
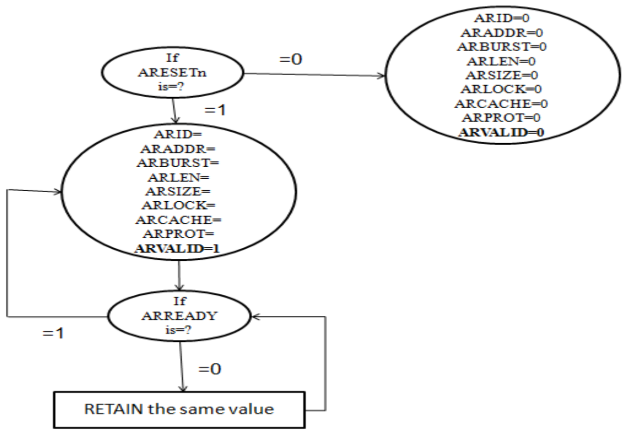


Fig -5: State diagram of Write Response Channel

3.1.4 Address Read Channel (AR Channel)

AXI_MASTER drives the command signals only when ARESETn is HIGH, else it drives all signals as zero. The address read command signals driven by the AXI_MASTER are - ARID, ARADDR, ARBURST, ARLEN, ARSIZE, ARCACHE, ARLOCK, ARPROT, with ARVALID as HIGH indicating that the driven signals are valid. The AXI_MASTER does not drive the ARVALID signal as LOW, until it receives the ARREADY signal, which is driven by the SOURCE_SLAVE, indicating that, it has received the address read command signals. If ARREADY is LOW, then AXI_MASTER retains the same values. Figure 6 shows the state diagram for the address read command signals.

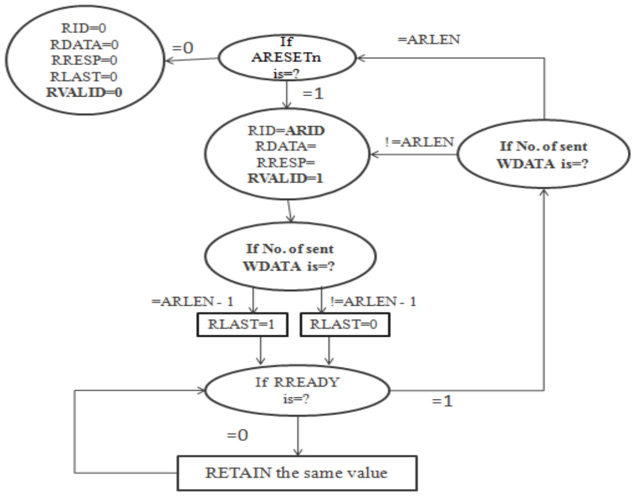


If ARESETn is=?
=0
ARID=0 ARADDR=0 ARBURST=0 ARLEN=0 ARSIZE=0 ARLOCK=0 ARCACHE=0 ARPROT=0 ARVALID=0
=1
ARID= ARADDR= ARBURST= ARLEN= ARSIZE= ARLOCK= ARCACHE= ARPROT= ARVALID=1
If ARREADY is=?
=1
=0
RETAIN the same value

Fig -6: State diagram of Address Read Channel

3.1.5 Read Data Channel (R Channel)

The SOURCE_SLAVE drives these Read Data signals after receiving the read command signals. It drives these signals, only when ARESETn is HIGH, otherwise it drives all signals as zero. SOURCE_SLAVE drives the RDATA signal with RVALID as HIGH, it holds the same value until it receives the RREADY signal. If RREADY is HIGH, it drives the next RDATA. SOURCE_SLAVE drives the ARLEN No. of data. While driving the last data it drives the RLAST as HIGH. Figure 7 shows the state diagram for the read data signals.



RID=0 RDATA=0 RRESP=0 RLAST=0 RVALID=0
=0
If ARESETn is=?
=ARLEN
=1
RID=ARID RDATA= RRESP= RVALID=1
!=ARLEN
If No. of sent WDATA is=?
If No. of sent WDATA is=?
=ARLEN - 1
!=ARLEN - 1
RLAST=1
RLAST=0
If RREADY is=?
=1
=0
RETAIN the same value

Fig -7: State diagram of Read Data Channel

3.2 Verification Environment of AXI Protocol

The verification environment for AXI bus is developed with SystemVerilog, this verification environment is shown in below Figure 8. This environment is organized in a hierarchical layered structure which helps to maintain and reuse it with different designs under verification.

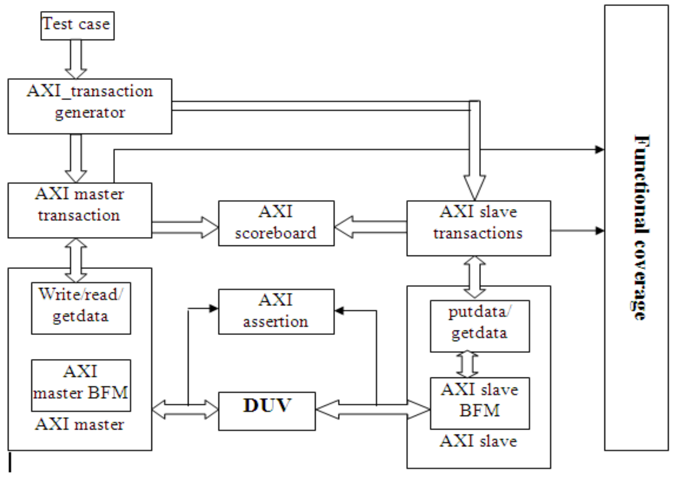


Fig -8: The Testbench Architecture

The main aim is to verify the design “AXI”, by applying different inputs.

3.2.1 Test Case

The 'Test case' includes the list of test cases. Each test case is connected to the "sequences" which written for the different scenarios like, single_write_operation, single_read_operation, write_followed_read_operation, multiple_write_single_read, single_wite_multiple_read, etc., Any one of the test case is connected to the Verification Environment to verify the design for a particular scenario.

3.2.2 AXI_Transaction_Generator

Transaction generator is also known as the "sequence item". Sequence_item is a class which includes all the port signals as its property. All these signals are declared using a "rand" keyword, so that after calling the randomize function this class should assign the random value to the each signal. This generated input values are assigned later to the DUV.

3.2.3 AXI_Master_Transaction

It includes the signals which are driven from the master. This class has the instance of the AXI_transaction_generator. The master transaction can override the values that are generated in the AXI_transaction_generator. Suppose we have not over ridden any signals, then the values that are generated in the AXI_transaction_generator are passed to the DUV.

3.2.4 AXI_Slave_Transaction

It includes the functionality similar to AXI_Master_Transaction, except it includes the signals which are driven from the slave.

3.2.5 AXI_Scoreboard

The values generated in the AXI_Master_Transaction and AXI_Slave_Transaction are also stored in the AXI_scoreboard. Later we can use these signals for the comparison of expected output and the actual output.

3.2.6 Functional Coverage

This class includes the list different coverage scenarios, which checks for the how much part of the design is covered during verification. AXI_Master_Transaction and AXI_Slave_Transaction classes will invoke this functional coverage.

3.2.7 AXI_Master

This is the main block of master part, it includes the two sub-blocks Write/read/get data and AXI master BFM.

Write/read/get data: This sub-block includes the objects of classes' sequencer, driver, and monitor. Sequencer picks the assigned sequence and drops it into the driver. It drives these signals according to the protocol. Monitor monitors whether signals are changing according to protocol or not

AXI master BFM: This is the class which includes the functions related to the buses. BFM stands for Bus Function Modules. Finally the signals driven from the driver are passed to the DUV.

AXI_Slave has the functionality similar to **AXI_Master.**

3.2.8 AXI_Assertions

It includes the list of assertions which are written according to the signal description. These are written using assert statements. These assertions are applied to the signals that are driving from the driver before applying to the DUV.

3.3 System Verilog

It is the Hardware Verification Language (HVL). This language is mainly used for the verification purpose. Initially, test bench (TB) is written in Verilog language using tasks and functions [11]. But it was a very lengthy process. It overcomes this lengthy process. System Verilog is the updated version of Verilog, it also supports the features like OOPs concept, Randomization and constrained randomization, etc., by the help of these features we can easily generate all the possible combinations of inputs, and thereby we can successively verify the Design.

4. CONCLUSIONS

AMBA AXI4 is a plug and play IP protocol. It is released by ARM, defines both bus specification and a technology independent methodology for designing, implementing and testing customized high-integration embedded interfaces. The data is 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. In this paper, an effective verification environment can simulate most cases of the AXI signal, check all the transmitted data automatically and complete coverage analysis during the simulation. So the environment can improve the coverage and reduce the time spending in the verification.

FUTURE SCOPE

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

ACKNOWLEDGEMENTS

I express my sincere gratitude to Dr. Udaykumar G. Khadke, Special Officer, Department of PG Studies, VTU, Gulbarga, I also thanks to Ms. Laxmi C for continuous guidance and other Professors of Department of VLSI Design and Embedded Systems, VTU, Gulbarga for extending their help & support in giving technical ideas about the paper without which I would not come up with this paper, I also thanks to all teaching, non teaching staff of Department of PG Studies, VTU, Gulbarga.

REFERENCES

[1]. 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.
[2]. Ref Shaila S Math, Manjula R B “Survey of system on chip buses based on industry standards,” Conference on Evolutionary Trends in Information Technology(CETIT), Belgaum, Karnataka, India, pp. 52, May 2011.
[3]. AMBA AXI Protocol Version: 2.0 Specification, ARM Ltd, pp. 1-1.
[4]. ARM, AMBA AXI protocol specifications, Available at, http://www.arm.com, 2003.
[5]. Silicore Corporation, Wishbone system-on-chip (soc) interconnection Architecture for portable IP cores.
[6]. IBM, Core connect bus architecture. IBM Microelectronics[Online].Available: http://www.ibm.com/chips/products/coreconnect, 2000.
[7]. M SivaPrasad Reddy, B. Babu Rajesh, Tvs Gowtham Prasad, “A Synthesizable Design of AMBA-AXI Protocol for SoC Integration,” in International Journal of Engineering Inventions Volume 1, Issue 3 (September2012) PP: 19-26.
[8]. 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.
[9]. L. Tao, X. Tong, Z. Yang, L. Huawei, and L. Xiaowei,”Bug analysis and corresponding error models in real designs”, in IEEE International High Level Design Validation and Test Workshop,2007,pp. 59-64.
[10]. Samir Palnitkar, Verilog HDL: A Guide to Digital Design and synthesis, 2nd ed, Hall PTR Pub, 2003.
[11]. C. Spear,” A Guide to Learning the Testbench Language Features”,in System Verilog for verification, 2nd ed., Springer Publishing Company, Incorporated,2008,pp. 11-18.