Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume 12, Issue 9, August 2021: 1259-1266

AMBA AXI4 Interconnect Universal Verification Component

Omkar Pataskar[a], Kshitij H. Gurjar[b*], Dr. Mrs. Vaishali Ingale[c] and
Ms. Ashlesha Gokhale[d]

[a,b,c]College of Engineering, Pune, India [d]ESCI-COMP Pvt. Ltd, Pune, India
[b]gurjarkh19.extc@coep.ac.in, [c]vvi.extc@coep.ac.in, [d]ashleshaag@gmail.com

Abstract

In today's Semiconductor Industry, at front-end level various IPs are designed separately and later to form a complex System on Chip, those are connected together to establish a proctocolitis communication. For such connection to form between various IPs, it becomes essential to design a interconnect which will follow all protocol rules and establish a correct information bypass. Advanced eXtensible Interface is such a type of protocol for microcontroller SoCs, primarily falling under AMBA family of ARM Holdings. All the data, address and response channels included in AXI are separate and independent. Once such a protocol design is done, there is extensive need of its verification by testing it in all the possible scenarios. The interconnect is quite crucial component when multiple masters and multiple slaves are used in a design environment. Interconnect makes sure that all the data coming from any of the masters is properly routed to the destined slave. The paper discusses about design of such a AMBA AXI4 based Interconnect UVC component which is used to verify a designed interconnect by giving various test scenarios alongside a multiple master and multiple slave environment.

Keywords: *UVM, UVC, AXI4, Interconnect, System Verilog, Verification, SoC*

1. Introduction

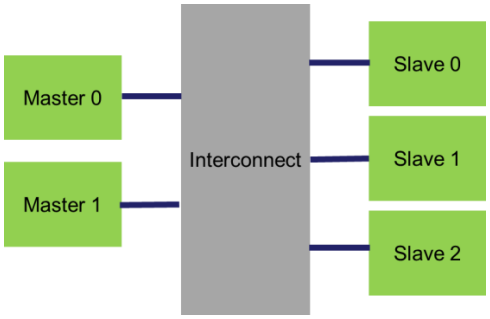
Today various different hardware components such as processor cores, peripherals, buses, controllers, bridges are the part of a wholesome system called as System on Chip (SoC). This SoC is to be used as a complete intelligent system with everything mounted on a single board. As per the Moore's law, we are decreasing the size of Chip as well as mounting numerous such components on that single board which in turn is increasing the complexity of entire system. While merging such diverse components, it becomes very crucial to maintain the form factor of the system. Rather than building separate components we can mount them on a single piece of board and connect them to form a fully functional protocol based communication system which exchanges data and information among various components of a system. As we are reducing the size of system, many more parameters such as area, size and performance are to be balanced and thus emerges the need of developing an efficient system which considers these parameters.

AMBA AXI4 protocol is a communication protocol used for systems containing uncached master and peripheral slave. AXI4 has three diverse channels which are READ, WRITE and RESPONSE, operating fully in lateral or parallel manner. The AXI4 Interconnect is the one sitting in between these masters and slave. The main job of this AXI4 Interconnect is to intelligently route the data and control information from one mastered side to other slave side. AXI4 Interconnect can do this routing among itself in parallel.

With the availability of parallel channels, it is easy to transfer information from one master1 to other slave1 whereas at same time we can transfer data from master2 to slave3 and many such scenarios can be formed by keeping in mind that the protocol violations are still preserved.

The purpose of this paper is to discuss the implementation of the AXI4 Interconnect as a verification component, where whole environment is written in SystemVerilog according to Universal Verification Methodology (UVM). The developed UVC component of AXI4 Interconnect contains many master many slave scenario where on the right side all the masters are interfaced and to the left side all the slave connections are being made. All the information flows through interconnect since it is not possible for a single master to communicate with different slaves at an intelligent rate. This verification IP is used to connect with the design hardware code written and further by applying randomly generated stimulus, the originally hard coded bugs in the design rtl file can be found out, Fig1. shows a block diagram of master slave architecture using interconnect.

Figure. 1. Many Master-Many Slave architecture



2. Methodology

The main functionality of this AMBA AXI4 Interconnect is to intelligently route the data and control information from one mastered side to other slave side where master can be uncached core and slave can be uncached peripheral device. Many master, many slave scenario is a bit more complex to design and thus it is more likely to have some bugs or defects embedded in system and thus can be found using verification IP component.

Certain research aspects of this paper includes developing UVC for AMBA AXI4 Interconnect using Universal Verification Methodology (UVM) and to verify a parallel many master, many slave communication over AXI4 channel. An separate AxPROT signal is being designed to deploy a priority to a particular master. Further to add an error dectection feature, an error feedback signal is introduced for incorrect slave address access attempt by the master by implementing a dummy slave inside interconnect. All the above features are embedded into the interconnect design which forms the part

of its methodology and then finally to Simulate design and see the waveforms on Cadence NCSim after giving random stimulus.

2.1 Universal Verification Methodology

Universal Verification Methodology is a IEEE standard methodology used for verification of RTL designs. Initially there were many EDA companies using their own verification language like Vera, VMM, OVM to verify the designs. But later only one methodology was accepted i.e. UVM. Main feature of UVM is its reusability where previous version of the verification component can be extended to form new version's component with added specifications. This actually saves time and complexity in verification and engineers can focus more on verification robustness.

The design which we want to verify is Design under Test DUT. The verification component is connected to this DUT via an interface and various test stimulus are provided to the DUT in the form of transactions or packets. The response from the DUT is further collected in monitor and is then cross verified with the golden responses/results stored in the reference design/model in the scoreboard.

2.2 Assumptions

The following assumption were made for developing the AXI Interconnect verification component:

a. ID size - 4bits

b. Size of Address - 32 bits

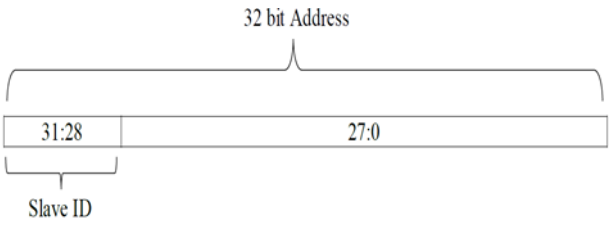
c. Size of Data – 32 bits

d. WSTROBE [Data Size % 8] = 4 bits

Table 1. Respective Slave Addresses

Slave ID	Initial Address	Final Address
0	0000_0000	0FFF_FFFF
1	1000_0000	1FFF_FFFF
2	2000_0000	2FFF_FFFF
3	3000_0000	3FFF_FFFF
4	4000_0000	4FFF_FFFF
5	5000_0000	5FFF_FFFF
6	6000_0000	6FFF_FFFF
7	7000_0000	7FFF_FFFF

Figure 2 : Assumptions for parametric lengths



2.3 UVC Design

All the mentioned design specifications are considered to develop the interconnect. Following is the functionality performed by the implemented design –

1. Active VALID bit detection
2. Calculate the respective PORT number where the communication is to be done
3. Communication occurs for the selected PORT
4. Wait if the other device is BUSY.
5. Bus arbitration feature is used if multiple master requests the same channel or same slave.

Interconnect is responsible for the information flow from master to slave devices connected. There is pre-defined address structure defined to each slave which allows to route the data properly through interconnect. During the build phase in UVM, all the defined address space is allocated to the connected slaves.

The memory for each master contains the following information –

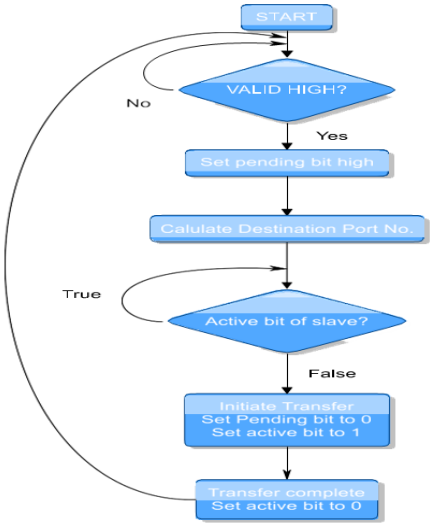
a. Port ID
b. Precedence ID prioritized
c. ACTIVE register [5-bits]
d. PENDING register [5-bits]
e. Write Address Target register
f. Read Address Target register

The slave part has following specification –

a. Port ID
b. Precedence ID prioritized
c. ACTIVE register [5-bits]
d. PENDING register [5-bits]
e. Target register

The working of the implemented design can be addressed as below using the flow chart shown in below figure. When the address of any slave is obtained, ACTIVE VALID bit needs to be snooped by the detector on all the active communication channels of AXI4.

Figure 3. Interconnect Flow diagram



After getting the VALID bit on the channel, the master which asserted the Valid, gets its PENDING register activated. As per the address available on the AXI4 interface channel, the target port of slave's address is generated. The interconnect then goes in standby mode where it is waiting for the ACTIVE register of the particular selected slave to get deactivated/cleared, which in turn depicts that the slave we selected is now ready to do the communication with the initiated master. All the information is transferred from active master to th selected slave via AXI4 channel. ACTIVE bit of master is asserted throughout the transactional process and deasserted once the communication is over with successful data transfer or reception.

3. Results

A. Many Master, Many Slave transaction

A condition where Many masters are trying to access diverse slaves at the same moment through AXI4 Interconnect is demonstrated here. The simulation of this condition is shown in following simulation console where M1 master and M0 master are making an attempt to communicate with S2 and S0 slave respectively. The simulation result shows that M1 and M0 both are able to complete their respective transfers successfully.

Figure 4. Multiple Master and Slave communication simultaneously

```
# UVM_INFO Expt12.sv(266) @ 7: uvm_test_top.env1.s2 [s2] RECEIVED ARVALID = 1
# UVM_INFO Expt12.sv(267) @ 7: uvm_test_top.env1.s2 [s2] RECEIVED ARID = 0x11
# UVM_INFO Expt12.sv(269) @ 7: uvm_test_top.env1.s2 [s2] RECEIVED ARADDR = 0xb
# UVM_INFO Expt12.sv(266) @ 7: uvm_test_top.env1.s0 [s0] RECEIVED ARVALID = 1
# UVM_INFO Expt12.sv(267) @ 7: uvm_test_top.env1.s0 [s0] RECEIVED ARID = 0x1
# UVM_INFO Expt12.sv(269) @ 7: uvm_test_top.env1.s0 [s0] RECEIVED ARADDR = 0x2
# UVM_INFO Expt12.sv(678) @ 7: uvm_test_top.env1.itc [itc] STARTING RDATA FOR MASTER  0
#
# UVM_INFO Expt12.sv(678) @ 7: uvm_test_top.env1.itc [itc] STARTING RDATA FOR MASTER  0
#
# UVM_INFO Expt12.sv(678) @ 7: uvm_test_top.env1.itc [itc] STARTING RDATA FOR MASTER  0
#
# UVM_INFO Expt12.sv(678) @ 7: uvm_test_top.env1.itc [itc] STARTING RDATA FOR MASTER  0
# UVM_INFO Expt12.sv(429) @ 11: uvm_test_top.env1.drv [drv] MASTER 1 RECEIVED DATA : 0xb
# UVM_INFO Expt12.sv(445) @ 11: uvm_test_top.env1.drv [drv] MASTER 0 RECEIVED DATA : 0x2
# UVM_INFO Expt12.sv(290) @ 11: uvm_test_top.env1.s2 [s2] TRANSMITTED RVALID = 1
# UVM_INFO Expt12.sv(291) @ 11: uvm_test_top.env1.s2 [s2] TRANSMITTED RDATA = 0xb
# UVM_INFO Expt12.sv(292) @ 11: uvm_test_top.env1.s2 [s2] WAITING FOR RREADY
# UVM_INFO Expt12.sv(290) @ 11: uvm_test_top.env1.s0 [s0] TRANSMITTED RVALID = 1
# UVM_INFO Expt12.sv(291) @ 11: uvm_test_top.env1.s0 [s0] TRANSMITTED RDATA = 0x2
# UVM_INFO Expt12.sv(292) @ 11: uvm_test_top.env1.s0 [s0] WAITING FOR RREADY
# UVM_INFO Expt12.sv(666) @ 13: uvm_test_top.env1.itc [itc] EXITING START ARADDR
#
# UVM_INFO Expt12.sv(666) @ 13: uvm_test_top.env1.itc [itc] EXITING START ARADDR
#
# UVM_INFO Expt12.sv(698) @ 15: uvm_test_top.env1.itc [itc] EXITING START RDATA
#
# UVM_INFO Expt12.sv(698) @ 15: uvm_test_top.env1.itc [itc] EXITING START RDATA
#
# UVM_INFO Expt12.sv(698) @ 15: uvm_test_top.env1.itc [itc] EXITING START RDATA
#
#
# UVM_INFO Expt12.sv(698) @ 17: uvm_test_top.env1.itc [itc] EXITING START RDATA
```

B. Many Master, Single Slave transaction

A condition where Many masters are trying to access a single slave at the same moment through AXI4 Interconnect is demonstrated here. As shown in following figure, the two masters MASTER 0 and MASTER 2 are stimulating the address *40000003h* and *40000045h* along with the active VALID bit at parallel moment to communicate with SLAVE port 4. In this case according to arbitration logic, the MASTER 0 is given access as the port number of it is LOWER over MASTER 2.

Figure 5**.** Priotizing the multiple accesses of masters to same slave

```
# UVM_INFO Expt13.sv(252) @ 40: uvm_test_top.itc.m [m] MASTER 2 ADDRESS GENREATED : 0x40000003
# UVM_INFO Expt13.sv(259) @ 40: uvm_test_top.itc.m [m] MASTER 0 ADDRESS GENREATED : 0x40000045
# UVM_INFO Expt13.sv(393) @ 40: uvm_test_top.itc.itc [itc] RECEIVED ARVALID FROM MASTER : 2
# UVM_INFO Expt13.sv(394) @ 40: uvm_test_top.itc.itc [itc] SETTING PENDING STATUS TO 1
# UVM_INFO Expt13.sv(393) @ 40: uvm_test_top.itc.itc [itc] RECEIVED ARVALID FROM MASTER : 0
# UVM_INFO Expt13.sv(394) @ 40: uvm_test_top.itc.itc [itc] SETTING PENDING STATUS TO 1
# UVM_INFO Expt13.sv(433) @ 41: uvm_test_top.itc.itc [itc] INITIATING TRANSFER FROM MASTER 0 (READ ADDRESS CHANNEL) TO SLAVE 4
# UVM_ERROR Expt13.sv(419) @ 45: uvm_test_top.itc.itc [itc] SLAVE BUSY : MASTER 2 WAS UNBALE TO ACCESS SLAVE 4 (READ ADDR CHANNEL)
# UVM_INFO Expt13.sv(136) @ 45: uvm_test_top.itc.s4 [s4] RECEIVED ARVALID = 1
# UVM_INFO Expt13.sv(137) @ 45: uvm_test_top.itc.s4 [s4] RECEIVED ARID = 0xx
# UVM_INFO Expt13.sv(139) @ 45: uvm_test_top.itc.s4 [s4] RECEIVED ARADDR = 0x40000045
# UVM_ERROR Expt13.sv(419) @ 47: uvm_test_top.itc.itc [itc] SLAVE BUSY : MASTER 2 WAS UNBALE TO ACCESS SLAVE 4 (READ ADDR CHANNEL)
# UVM_ERROR Expt13.sv(419) @ 49: uvm_test_top.itc.itc [itc] SLAVE BUSY : MASTER 2 WAS UNBALE TO ACCESS SLAVE 4 (READ ADDR CHANNEL)
# UVM_INFO Expt13.sv(447) @ 51: uvm_test_top.itc.itc [itc] TRANSFER COMPLETED FROM MASTER 0 (READ ADDRESS CHANNEL) TO SLAVE 4
# UVM_INFO Expt13.sv(433) @ 51: uvm_test_top.itc.itc [itc] INITIATING TRANSFER FROM MASTER 2 (READ ADDRESS CHANNEL) TO SLAVE 4
# UVM_INFO Expt13.sv(136) @ 53: uvm_test_top.itc.s4 [s4] RECEIVED ARVALID = 1
# UVM_INFO Expt13.sv(137) @ 53: uvm_test_top.itc.s4 [s4] RECEIVED ARID = 0xx
# UVM_INFO Expt13.sv(139) @ 53: uvm_test_top.itc.s4 [s4] RECEIVED ARADDR = 0x40000003
# UVM_INFO Expt13.sv(447) @ 59: uvm_test_top.itc.itc [itc] TRANSFER COMPLETED FROM MASTER 2 (READ ADDRESS CHANNEL) TO SLAVE 4
```

The above mentioned functionality is verified in in simulation which is depicted in above figure.

C. Dummy slave active when incorrect address stimulus

AXI4 specification has AxPROT signal which gives us DECERR response when an incorrect slave access is initiated. This response is generated from a dummy slave residing inside incorrect. Whenever a write instruction is generated by master with an incorrect address, the response is given by dummy slave on Write Response channel whereas for a read instruction generated, the response is thrown on a Read Data channel of AXI4 interface. Incorrect address which is actually outside the address space of the implemented slaves, is given as an error to master by interconnect.

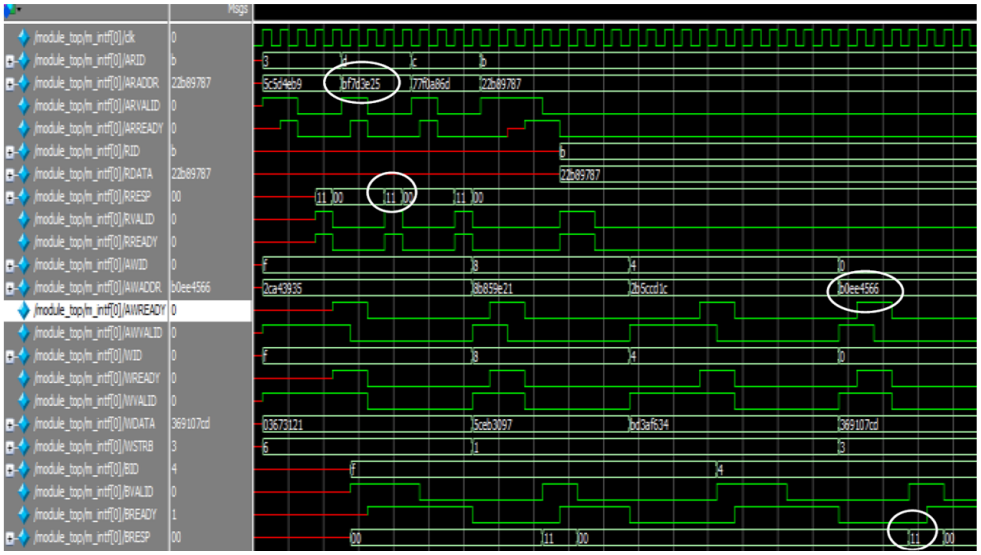
Figure 6. Dummy slave coming in action when incorrect address stimulus generated

```
master_driver_write.svh(51) @ 50: uvm_test_top.env.axi_master_agent0.master_driver_w [master_driver_w] MASTER 0 SENDING WRITE ADDRESS 0x25fb2c2f
master_driver_read.svh(47) @ 50: uvm_test_top.env.axi_master_agent0.master_driver_r [master_driver_r] MASTER 0 SENDING READ ADDRESS 0x78713416
master_driver_write.svh(51) @ 50: uvm_test_top.env.axi_master_agent1.master_driver_w [master_driver_w] MASTER 1 SENDING WRITE ADDRESS 0x44f86d54
master_driver_read.svh(47) @ 50: uvm_test_top.env.axi_master_agent1.master_driver_r [master_driver_r] MASTER 1 SENDING READ ADDRESS 0x2f3686c5
interconnect.svh(134) @ 50: uvm_test_top.env.intc [intc] RECEIVED INVALID READ ADDRESS FROM MASTER : 0. ACCESS TO SLAVE 7 IS INVALID
interconnect.svh(455) @ 55: uvm_test_top.env.intc [intc] EXECUTING DEFAULT SLAVE FOR RESPONSE TO MASTER 0
interconnect.svh(455) @ 65: uvm_test_top.env.intc [intc] EXECUTING DEFAULT SLAVE FOR RESPONSE TO MASTER 1
master_driver_read.svh(72) @ 85: uvm_test_top.env.axi_master_agent0.master_driver_r [master_driver_r] MASTER 0 RECEIVED ERROR RESPONSE FOR READ ADDRESS 0x78713416. RECEIVED DATA IS INVALIDATED
master_driver_read.svh(47) @ 90: uvm_test_top.env.axi_master_agent0.master_driver_r [master_driver_r] MASTER 0 SENDING READ ADDRESS 0x52175f53
interconnect.svh(134) @ 90: uvm_test_top.env.intc [intc] RECEIVED INVALID READ ADDRESS FROM MASTER : 0. ACCESS TO SLAVE 5 IS INVALID
interconnect.svh(455) @ 95: uvm_test_top.env.intc [intc] EXECUTING DEFAULT SLAVE FOR RESPONSE TO MASTER 0
interconnect.svh(328) @ 115: uvm_test_top.env.intc [intc] TRANSFER COMPLETED FROM MASTER 1 (READ ADDRESS CHANNEL) TO SLAVE 2
master_driver_read.svh(72) @ 125: uvm_test_top.env.axi_master_agent0.master_driver_r [master_driver_r] MASTER 0 RECEIVED ERROR RESPONSE FOR READ ADDRESS 0x52175f53. RECEIVED DATA IS INVALIDATED
master_driver_read.svh(47) @ 130: uvm_test_top.env.axi_master_agent0.master_driver_r [master_driver_r] MASTER 0 SENDING READ ADDRESS 0x5e7a0cc4
interconnect.svh(134) @ 130: uvm_test_top.env.intc [intc] RECEIVED INVALID READ ADDRESS FROM MASTER : 0. ACCESS TO SLAVE 5 IS INVALID
interconnect.svh(455) @ 135: uvm_test_top.env.intc [intc] EXECUTING DEFAULT SLAVE FOR RESPONSE TO MASTER 0
master_driver_read.svh(74) @ 135: uvm_test_top.env.axi_master_agent1.master_driver_r [master_driver_r] MASTER 1 RECEIVED RESPONSE FOR READ ADDRESS 0x2f3686c5 WITH DATA 0x2f3686c5
master_driver_read.svh(47) @ 140: uvm_test_top.env.axi_master_agent1.master_driver_r [master_driver_r] MASTER 1 SENDING READ ADDRESS 0x97021e53
interconnect.svh(134) @ 140: uvm_test_top.env.intc [intc] RECEIVED INVALID READ ADDRESS FROM MASTER : 1. ACCESS TO SLAVE 9 IS INVALID
interconnect.svh(455) @ 145: uvm_test_top.env.intc [intc] EXECUTING DEFAULT SLAVE FOR RESPONSE TO MASTER 1
master_driver_write.svh(102) @ 145: uvm_test_top.env.axi_master_agent1.master_driver_w [master_driver_w] MASTER 1 RECEIVED RESPONSE FOR WRITE ADDRESS 0x44f86d54
```

D. Simulation Waveform :

The simulation was made possible on the Mentor's QuestaSim Verification Software 10.4e. The different stimulus are provided by a random address and data generator and further interconnect break down this information to identify the slave to which communication is requested by master. The correctness of the signals in waveform can be observed by identifying read, write signals both on master as well as slave side interfaces. Fig 7 shows signals on the interface after generation of an incorrect and out of the address region signal. When no particular slave is confined to a particular start address, the DECERR error is thrown on the response channel with value as 0x11. Fig6 shows MASTER 1 and SLAVES S1, S2, S3 and S4 are to be active. So, the random address generated greater than 0xd will be considered as ERROR by interconnect. When read address ARADDR is 0x**bf7d3e25**, that is an incorrect address, the signal RRESP becomes 0x11 after 3 clock cycles thus concluding that randomly generated address 0x**bf7d3e25** incorrect. Likewise, when the write address AWADDR is 0x**b0ee4566** which is an incorrect address, the signal BRESP become 0x11 after three clock cycles indicating that the generated address is invalid.

Figure 7**.** Interface signals for Invalid address (MASTER 0)



4. Conclusions

We could develop the UVC based Interconnect on AXI4 protocol successfully with SystemVerilog using UVM. The Interconnect UVC is able to handle diverse three conditions of many master-many slave, many master-single slave and error response through dummy slave.

Various randomly generated addresses and control signals were given to the interconnect to cross verify the functionality of successful connection between master requesting a particular slave by its addresses. This UVC can be then connected to the Master and Slave SoC based designs.

References :

[1] M. R. Nigam and S. Bande, "AXI Interconnect Between Four Master and Four Slave Interfaces", International Journal of Engineering Research and General Science, 2014.

[2] K. Salah, "A UVM-Based Smart Functional Verification Platform: Concepts,Pros,Cons and Opportunities", IEEE 9th International Design and Test Symposium, 2014.

[3] C. W. Wang, “On-chip Interconnection Design and SoC Integration with OCP”, IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT), 2008.
[4] ARM, AMBA® AXI and ACE Protocol Specification, England, 2017.
[5] S. G. Mahesh, “Verification of Memory Transactions in AXI Protocol using System Verilog Approach”, IEEE International Conference on Communication and Signal Processing, 2015.
[6] Xilinx, AXI Interconnect v2.1 LogiCORE IP Product Guide, 2017.
[7] Accellera, Universal Verification Methodology (UVM) 1.2 Users Guide.
[8] C. Spears, System Verilog for Verification : A guide to learning testbench language features, Springer Publications.