Verification of Advanced High Performance Bus Arbiter using System Verilog Assertion

Prince Gurha[1], Dr.(Mrs.) R. R. Khandelwal[2]
Dept. of Electronics
R.C.O.E.M, Nagpur

Abstract- *With the fanatically improvement in electronics interconnection technologies, Arbiter is required for high performance buses. Arbiters subsist in almost every logic design. In many designs a large number of requesters must access a same resource. An arbiter is used to arbitrate how the resource is shared amongst the multiple requesters. The arbitration plays a captious role to determine the performance of bus based system, as it assign precedence with which processors is grant the access to the shared communication resources. In this paper, Round robin arbitration is used for scheduling and verified by using (SVA) SystemVerilog Assertion as it is widely used verification techniques to enhance the verification quality and reduce the debugging time of complex designs in order to speedup the verification process. Here, Advanced High Performance Bus (AHB) Arbiter is designed in verilog language. This design is then verified using SVA binding construct in ModelSim. Binding allows verification engineers to add assertions to design without touching the design files.*

Keywords- *Verification, Arbiter, SVA, Bind, Assertion.*

I. INTRODUCTION

Arbiter is the main controlling component in the design. It is used to grant access to a individual bus master at a time[1]. Every bus master has a Req or Gnt signal to the bus arbiter. The limit of the arbitration scheme may decline the system performance because arbitration scheme depends on the application demand. By employing an efficient arbitration scheme, the design can be used to aid for better applications. The arbiter use anyone of the arbitration algorithm which decides the priority of the master likes Round robin Algorithm, Priority Scheduling Algorithm, Fixed Priority Algorithm etc. Here in this project we are using Round Robin Algorithm for the designing of Arbiter. Round robin arbitration is generally used for scheduling. [2].

A round robin arbiter ensures competence among masters and allows vacant time slot to be shared to a master whose round robin turn is later but who is ready now. In round robin arbitrations mechanism, the accesses to the Bus is given to the masters as per there sequence. Round robin arbiter gives 1st priority to the 1st master, 2nd to the 2nd masters and so on[2]. Arbiters exist in nearly every logic designs and need to be verified to achieve high performance[1]. But Nowaday's Verification is very challenging task for the designer in the entire design and verification cycle, because bugs which are uncovered in the earlier stages of the design carry on in the next stages of the design and later it is too complex to diagnose it. Verification is a task of verifying whether our implementation matches with the micro-architectural specifications or not. Any design is incomplete without proper verification of that design[3].

The time required for verifying the design is becoming monotonous as the complicacy of the chip design is increasing exponentially. Nowadays, about 70% of the design time is needed for developing the verification environment. Hence in this condition, it is important to reduce the verification time in order to speed up the whole development process[3]. Therefore, to improve the design observability and to detect its faults, Assertion Based Verification (ABV) is introduced. ABV is a technique in which assertions are used to grab specific design behavior either through simulation, formal verification or emulation of these assertions[4].

SVA is a type of ABV. SystemVerilog Assertion is a formal verification language. It is also an integral part of SystemVerilog. SVA is a declarative language which gives excellent control over time. The language itself is very concise and is very easy to maintain[5]. SVA cannot be written directly into Hardware Description Languages other than SystemVerilog, but tool support for the use of SVA with other HDLs is possible via binding directives and comment pragmas[5].

Deploying assertions has several advantages which can be summarized as below: SVA can easily be disabled or enabled at any point during simulation, as needed[4]. SVA is not only used to debug pre-silicon violations, but is also extended to debug post-silicon violations. Uses only concurrent assertion. In a concurrent assertion evaluation is performed only at the occurrence of a clock tick. It is also used for the creation of complex properties[4].

The paper is presented as follow: Section II and III introduces the overall architecture of AMBA-AHB and its applications, Section IV gives detail

about SVA building block, its binding properties and also gives detail about property description of Arbiter using SVA properties, Section V gives Simulation results of Arbiter while Section VII concludes the paper.

II. ARCHITECTURE OF AMBA-AHB

The ARM Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip communication protocol for the connection and management of different blocks in system-on-a-chip (SoC) designs. The (AHB) Advanced High Performance Bus is used to achieve high performance and high bandwidth operations. It can support 16 masters and slaves. It consist of masters, slaves, an arbiter and an address decoder. The address bus can be upto 32 bit wide and the data bus can be upto 128 bit wide. There is a single global clock[3].

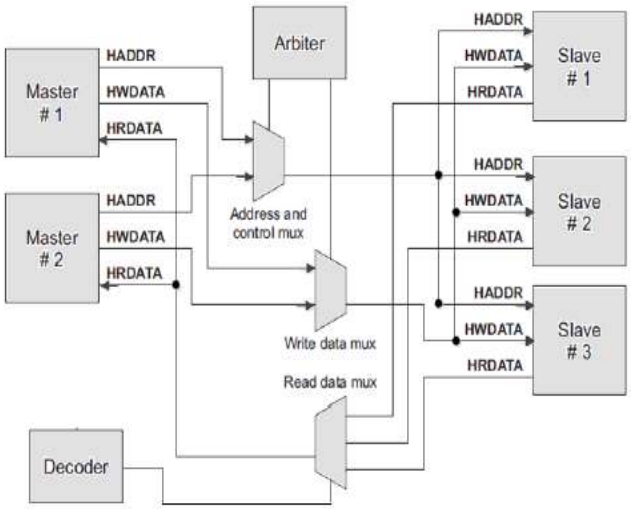

Fig. 1 Architecture of the AMBA AHB

Fig. 2 shows high-performance bus which sustain the external memory bandwidth, on which the high bandwidth external memory, on chip Memory and other Direct Memory Access (DMA) devices situated. It also provides a high bandwidth interface between the different components that are involved in the bulk of transfers[3].

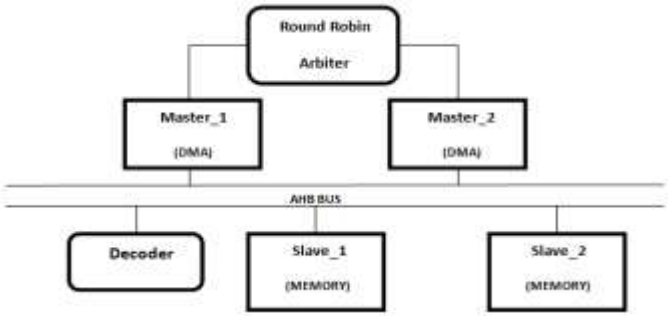

Fig. 2 Typical AMBA Based Microcontroller

- The basic transfer consists:

1) HCLK: global clock signal.
2) HWDATA: write data from Master to Slave.
3) HRDATA: read data from Slave to Master.
4) HREADY: Ack signal from Slave to Master.
5) HADDR: target address for transfer[6].

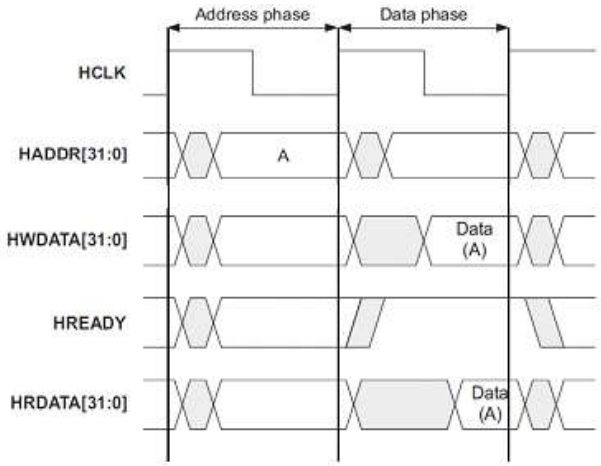

Fig. 3 Basic transfer

Signals from Master to Slave involves: With reference to posedge of clock, Master X is putting its address and control signals. If it is write operation, then data corresponds to first address comes to second clock cycle. Simultaneously, the address for next location is coming. This is a pipelining operation. Instead of wasting the address bus we are utilizing the address bus. Therefore, data for the second address comes in next clock cycle. The HREADY is a "ack" signal from the slave. If it is ready to receive data it makes ready signal high. If it is not ready to receive data it makes ready signal low, so master will retain the same address and data. For read operation, data corresponds to first address comes to second clock cycle and read data from slave to master whenever HREADY is high.

III. COMPONENTS OF AMBA AHB

The main components of AMBA-AHB are:

A. AHB Master- It is used for read and write operations by providing address and control information. At one time only single master can access the bus. To start a transfer, it drives its bus request signal high[1].

B. AHB Slave- It is used for performing read and write operations. Slave determines how the transfer should process after master has started a transfer. Whenever the slave is accessed it must provide acknowledgement which denotes the status of the transfer[1].

C. AHB Decoder- The decoder decodes the address given by the master and selects the slave for the

data transfer process. The AHB decoder simply perform a direct decode of the address bus[1].

D. AHB Arbiter- Arbiter is the main controlling component in the design. It is used to grant access to a individual bus master at a time. Every bus master has a Req or Gnt signal to the bus arbiter[1].

- Advantages of round robin algorithm are:
  1) Its architecture is simple.
  2) No interrupt and no shared data.
  3) No response latency[7].

- Three steps of Arbitration:

***1) Step 1*:** Request.

Every input signal sends a request to every output for which it has a queued cell.

2) Step 2: Grant.

If an output gets any requests, it selects the one that occurs next in a fixed, round robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted.

3) Step 3: Accept.

Step 3: Accept. If an input receives a grant, it accepts the one that occurs next in a fixed, round-robin schedule starting from the highest priority element[7].

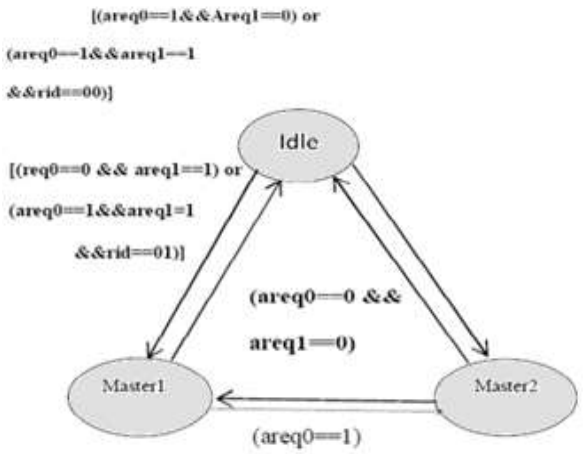

Fig. 4 State diagram for AHB- Arbiter

- Areq_0 & Areq_1 are the requests generated from master_1 & master_2 respectively.
- If master_1 requests for slave access then Areq_0 is set to one & Areq_1 signal is forced to be in idle state. State transition of master_1 takes place from idle to master_1 state.
- After completion, transfer of data in case Areq_1 is set then the transition takes from master_1 to master_2. If Areq_1 is not set then the transition occurs from master_1 to idle state.
- If master_2 requests for slave access then Areq_1 is set to one & Areq_0 signal is forced to be in idle state. State transition of master_2 takes place from idle to master_2 state.
- After completion, transfer of data in case Areq_0 is set then the transition takes from master_2 to master_1. If Areq_0 is not set then transition occurs from master_2 to idle state.
- If both master's requests are set i.e, Areq_0=1 && Areq_1=1 then it depends on the internal signal rid. If rid signal is set 00 then master_1 gets the access. If rid value is set to 01 then master_2 gets the access[2].

IV. VERIFICATION PROCESS

In any design model, the functionality is represented by the combination of multiple logical events. These logical events are simple boolean expressions which is assessed on the same clock edge or events that get assessed over a period of time involving multiple clock cycles [8].

The process of verification follows these steps:

***1) Design of RTL Module*:** Here, AHB-Arbiter is designed in verilog language.

2) Formation of SVA Properties:

assertion property:

assert property (@(*sample_signal*) disable iff (*expression*)
// *optional disable condition*
property_expression_or_sequence);

assert property - keywords to start the definition of an assertion.
(...); - placeholder for the assertion.
@(posedge clk) – sample clock signal for concurrent assertion.
disable iff (!rst_n) – to disable the assertions.
property*_expression_*or_sequence - actual assertion test[8].

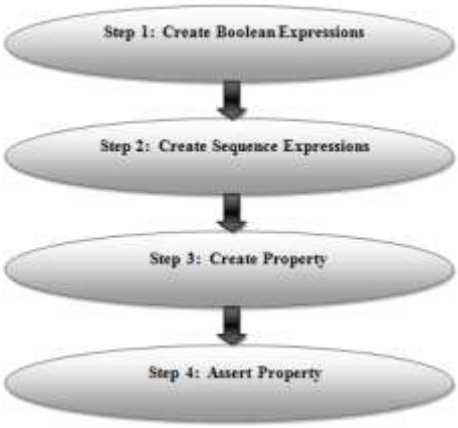

Fig. 5 SVA building block

- ***Property Description of AHB-Arbiter using SVA properties:*** Based on the property description of the AHB-Arbiter, the following SVA checks can be extracted. Creating such intermediate expressions makes the SVA checkers more readable.

CHECK #1:

```
property checkForValidGrant; @(posedge clk)
   disable iff ( ! reset)
        sx  |=> sy endproperty
sequence sx; (hmaster==4'b0000) ; endsequence
sequence sy; (s_grant0 == 1'b0 ##1 s_grant1 == 1'b0); endsequence
assert property(checkForValidGrant) else $error("Invalid Grant Signal Given");
```

CHECK #2:

```
property INVALID_GRANTS; @(posedge clk)
   disable iff ( ! reset)
        sx  |=> sy endproperty
sequence sx; (s_grant0) ; endsequence
sequence sy; (s_grant1); endsequence
assert property( ! INVALID_GRANTS );
```

CHECK #3:

```
property INVALID_HMASTER; @(posedge clk)
    disable iff ( ! reset)
         sx  |=> sy endproperty
sequence sx; (hamster) ; endsequence
sequence sy; (4'b0000); endsequence
assert property( ! INVALID_HMASTER );
```

CHECK #4:

```
property SUCCESS_VALID_GRANT;
  @(posedge clk) disable iff ( ! reset)
       ((s_grant0 == 1'b0 && s_grant1==1'b1)  ##1 (s_grant0 == 1'b1 && s_grant1==1'b0));
endproperty
assert property(SUCCESS_VALID_GRANT);
```

3) Connecting RTL Module to the Property Module:

SVA can be connected to the design by different methods:

- Adding the assertions directly into the RTL source code.
- Placing the assertions into a separate module (separate file) and add the module into the RTL source code using the **`include** compiler directive.
- Placing the assertions into a separate module (separate file) and binding the module to the design from a third module, such as a testbench[8].

- ***SVA Binding Construct:***

To make verification separate from design, it is possible to define properties and bind them to specific modules or instances. It permits verification engineers to verify with minimal changes to the design code. No semantic changes to the assertions are used due to this feature[9].

Example of binding two modules:

```
module dma ( a, b, c) // module containing the design
< RTL Code >
endmodule
module dma_property ( a, b, c) //module containing properties
< Assertion Properties >
endmodule
Bind dma dma_property dma_inst_1(a, b, c); //module that binds design module to property module.
```

- dma and dma_property are the module name.
- dma_inst_1 is dma_property instance name.
- Ports (a, b, c) gets bound to the signals (a, b, c) of the module dma[10].

V. ASSERTION BASED VERIFICATION OF ARBITER USING DIFFERENT SVA PROPERTIES:

Assertion Based Verification of AHB-Arbiter Using Different SVA Properties: The different properties of AHB-Arbiter are verified by writing assertion for the properties and verified it using ModelSim. The following table describes some of the SVA properties that are verified for the AHB-Arbiter design. The AHB-Arbiter is verified by 10 properties which are described below:

TABLE I. VERIFICATION OF AHB-ARBITER

S.No.	PROPERTY DESCRIPTION	STATUS
1.	Master_1 sends a request signal to the arbiter.	PASS

2.	Master_2 sends a request signal to the arbiter.	PASS
3.	Master_3 sends a request signal to the arbiter.	PASS
4.	Arbiter gives a grant signal to the master_1.	PASS
5.	Arbiter gives a grant signal to the master_2.	PASS
6.	Arbiter gives a grant signal to the master_2.	PASS
7.	Arbiter is in idle state.	FAIL
8.	Master_1 first sends a request signal to the arbiter, but arbiter gives the grant signal to the master_2.	FAIL
9.	Master_2 first sends a request signal to the arbiter, but arbiter gives the grant signal to the master_3.	FAIL
10.	Master_3 first sends a request signal to the arbiter, but arbiter gives the grant signal to the master_1.	FAIL

VI. SIMULATION RESULTS OF ARBITER

Here in our case we are using round- robin algorithm for designing of arbiter. A round robin arbiter ensures competence among masters and allows vacant time slot to be shared to a master whose round robin turn is later but who is ready now. This simulation result shows that at any one clock cycle only one request can access the grant signal. As shown in the figure, the bus request signal for the master1 goes to high the grant signal for the master1 is high.

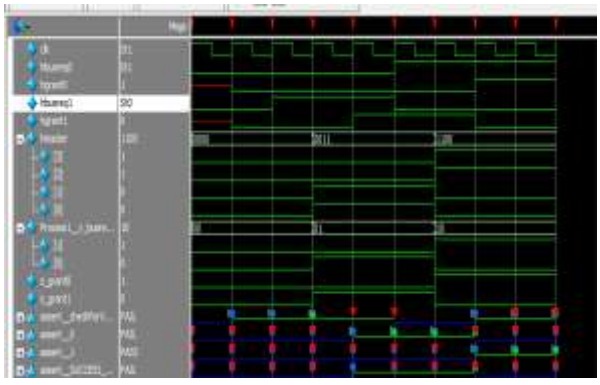

Fig. 6(a) Simulation of Arbiter

If a design has been compiled with assertion, we can view the assertion waveform in wave window. As shown in the fig. 6 (b). The left column of the figure shows the name of assertion directive and the name of each directive comes from assertion code.

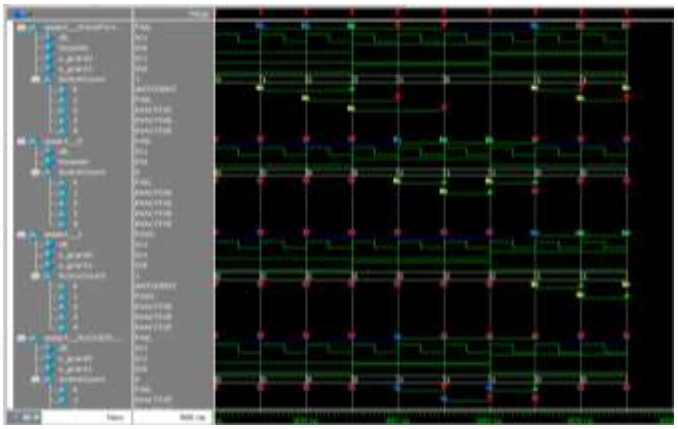

Fig. 6(b) Simulation of Arbiter (Assertions)

TABLE II. ASSERTION REPORT

Assertion name	Assertion type	Language	Enable	Fail count	Pass count	Active count	Peak memo	Peak Memo time	Cumulative threads
Check for Valid Gnt	concurrent	SVA	on	4	1	1	1448	300 ns	6
Invalid Hmaster	concurrent	SVA	on	7	3	0	968	500 ns	10
Invalid Grant	concurrent	SVA	on	7	2	1	968	800 ns	10
Success Valid Gnt	concurrent	SVA	on	9	1	0	968	500 ns	10

TABLE III. DESCRIPTION OF ASSERTIONS

Graphic Element	Meaning
blue line	assertion or cover directive is inactive
green line	assertion or cover directive is active
blue square	assertion or cover directive starts
green triangle	assertion or cover directive passed
red triangle	assertion or cover directive failed
yellow triangle	antecedent match occurred in assertion

VII. CONCLUSION

SystemVerilog Assertion (SVA) ensure true assertions based verification integrated into Verilog/SV language. SystemVerilog Assertion can easily be turned ON or OFF at any instant during simulation as needed. Assertion have full visibility to all design code, don't have to hide in comments as in PSL. Easy to write as compared to other solutions. Binding allows verification engineers to add assertions to design without touching the design files. SVA are a team effort some assertions are written by the design team and some are written by the verification team. We design the AHB-Arbiter in Verilog language and verified using SVA bind construct in ModelSim.

REFERENCES

[1] Divekar, Shraddha, and Archana Tiwari. "Interconnect matrix for multichannel AMBA AHB with multiple arbitration technique",

2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), 2014.
[2] M. Conti, M. Caldari, G.B. Vece, S. Orcioni, C. Turchetti, "Performance Analysis of Different Arbitration Algorithms of the AMBA AHB Bus", in Proc. of IEEE Conference on Design Automation , 2004, San Diego, USA, pp. 618 – 621.
[3] Akshay Mann, Ashwani Kumar," Assertion Based Verification of AMBA-AHB Using Synopsys VCS®"International Journal of Scientific & Engineering Research, Volume 4, Issue 11, November-2013 ISSN 2229-5518).
[4] Mostafa, Moaz, Mona Safar, M. Watheq El-Kharashi, and Mohamed Dessouky. "System Verilog Assertion Debugging Based on Visualization, Simulation Results, and Mutation", 2014 15th International Microprocessor Test and Verification Workshop, 2014.
[5] PSL AND SVA: TWO STANDARD ASSERTION LANGUAGES ADDRESSING COMPLEMENTARY ENGINEERING NEEDS John Havlicek, Freescale Semiconductor, Inc., Austin, TX Yaron Wolfsthal, IBM Haifa Research Lab, Haifa, Israel.
[6] M.J. Dougherty. "A SEM-E module avionics computer with PI-Bus backplane communication", IEEE Conference on Aerospace and Electronics, 1990.
[7] International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463 Vol. 2 Issue 9, September-2013, Speed efficient implementation of round robin arbiter design using VERILOG Ruma Deb, Dr Rajrajan ,Dept. of Electronics & Communication Engg., Sathyabama University, Chennai (TN), India
[8] Ashok B. Mehta Los Gatos, CA USASystemVerilog Assertions and Functional Coverage, Guide to Language, Methodology and Applications, ISBN 978-1-4614-7323-7 ,DOI 10.1007/978-1-4614-7324-4 Springer New York Heidelberg Dordrecht London.
[9] "Introduction to SVA", A Practical Guide for SystemVerilog Assertions, 2005, Microprocessor Systems, 1995 & Understanding Behavioral Synthesis, 1999 & ARM Ltd., AMBA specification (rev. 2) 1999.
[10] SystemVerilog Assertions and Functional Coverage,2014.