VERIFICATION OF AHB PROTOCOL USING SYSTEM VERILOG ASSERTIONS

Nikhil B. Gaikwad[1], Vijay N. Patil[2]

[1] *P.G. Student, Electronics & Telecommunication Department, Pimpri Chinchwad College of Engineering, Pune, Maharashtra, India.*
[2] *Professor, Electronics & Telecommunication Department, Pimpri Chinchwad College of Engineering, Pune , Maharashtra, India.*

ABSTRACT

Increasing technology increases the amount of logic that can be placed in a silicon chip driving highly integrated SoC design development. The most relevant factor for a SoC is how well they are interconnected. The SoC uses an on-chip bus architecture. AMBA (Advanced Microcontroller Bus Architecture) bus is most widely used the on-chip bus which is introduced by the ARM. Almost 85-90% of on-chip bus used in any SoC is AMBA (Advanced Microcontroller Bus Architecture). In this paper, an AMBA AHB bus Verification environment is built which is verified by using System Verilog Assertions. AMBA protocol (AHB) is verified by achieving successful read & write operations for incrementing burst feature. Simulation of AHB verification environment is done in Questa sim-simulator tool (from Mentor Graphics).

Keyword : - *AMBA ,AHB , SV, SVA .*

1. INTRODUCTION

In the modern years, due to the increasing market condition for low power, min area, minimum engineering cost and high performance systems as well as the enhancement of the semiconductor process technology, Very Large Scale Integration (VLSI) demand increased to an extent where all the system components are required to integrated on a single chip called system on chip[2]. These intellectual property (IPs) with different functionality are integrated within a chip, and maybe all these IP modules had completed their design and verification independently. However, some scenarios get raised unexpected with the individual IP's and hence the whole SoC will lead to failure. Incompatibilities between the IP interfaces cause common problem with transaction error. Thus for implementing any System on chip(SoC) it has become rule that standardized and preverified bus protocol interface architecture should be integrated. So on-chip communication using a bus protocol, whose specification provides a common interface that facilitates IPs integration became the basis of SoC architecture. To promote reusability of IPs and to get the SoC integration within time, some of the standards are set to bus-based communication architecture over the past several years..

During the development of these kinds of SOC's, verification becomes very a pretty time consuming and challenging task. From engineering research it is clear that 70% time of overall project development cycle is generally estimated to be consumed by verification compared with design stage
which requires only 30% out of the total time. Hence, number of verification engineers are required for verifying an on-chip communication protocol and their functional properties and the synchronization between the IP's within the SoC, with the help of Verification IP having dedicated verification environment. Communication bus protocols used in today's SOC's are basically classified on the basis of their performance, silicon area and power consumption. Among the three AMBA protocols APB bus protocol structure is simplest one to design and requires less power compare to AHB bus protocol structure but it has low performance than AXI Bus[4]. The AMBA AXI power consumption is at moderate level and gives advances in performances than AHB and APB. AHB architecture is of type the shared bus, so arbitration technique used to grant the access of bus to only one master at a time. Final choice to use protocol is is based on the utilization factor in the SoC design and the capability of the interconnected peripheral modules. In this paper we discuss the design of a System Verilog

Assertion (SVA) based Verification for verifying AMBA AHB protocol using a functionality checks driven methodology.

2. ARCHITECTURE OVERVIEW

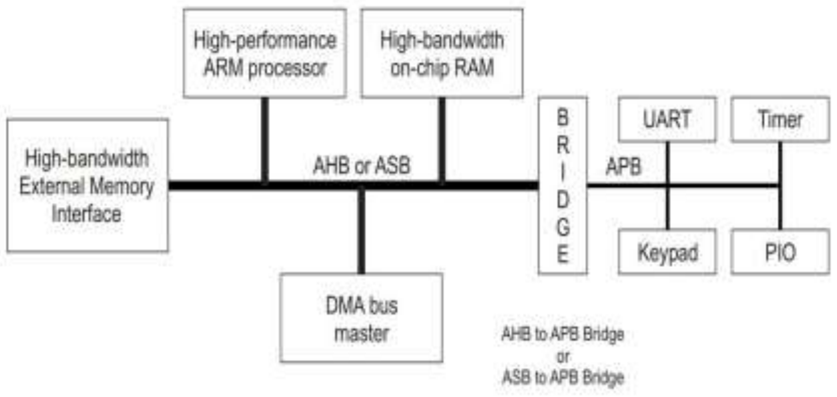

High-performance ARM processor
High-bandwidth on-chip RAM
High-bandwidth External Memory Interface
AHB or ASB
BRIDGE
APB
UART
Timer
Keypad
PIO
DMA bus master
AHB to APB Bridge or ASB to APB Bridge

Fig -1: General ARM Architecture [1]

An ARM architecture based microcontroller system consist of verity of AMBA protocols (APB, AHB, ASB or AXI) as per the system requirement. Basic function of the AMBA protocol is to provide communication media for the peripheral devices. SoC consists of high bandwidth memory, on chip memory and Direct memory Access Device. These communication protocols have to provide high bandwidth interface for these memory to improve the SoC performance hence use of high performance bus (AHB, AXI) is required. If the system contain peripherals of high as well low performance type, then along with high performance AHB, AXI bus protocol use of low performance APB is also there. When system requires both high and low performance at the same time then bridge concept comes in the picture.

2.1 Description of AMBA AHB Protocol:

Protocol is nothing but set of rules like how to issue a request, how to write the data, how to read the data, how to drive the response etc. ARM introduces on-chip protocols AHB, APB comes from AMBA 3 family. AXI comes from AMBA 4 family. Below is the description of AHB and AXI protocol. Basically these protocols are differ from each other depends on their performance and performance is nothing but how much data is transferred at a time (in a single cycle).APB is used for communication between low performance devices, AHB is used for communication between high performance devices, AXI is used for communication between very high performance devices with overall high system level throughput requirement. The Advanced Microcontroller Bus Architecture (AMBA) specification defines an on chip communications standard for designing high-performance embedded microcontrollers.

Three distinct types of bus protocols are defined within the AMBA specification:

- The Advanced High-performance Bus (AHB)
- The Advanced System Bus (ASB)
- The Advanced Peripheral Bus (APB).

A test methodology is included with the AMBA specification which provides an infrastructure for modular macrocell test and diagnostic access.

2.2 Advanced High-performance Bus (AHB):

The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques [1].

2.2.1 AMBA AHB Component:

A general AMBA AHB subsystem design consists of following components:

- **AHB Master**

A bus master is able to initiate read and write operations by providing an address and control information. Only one bus master is granted with the access to use the bus at a time.

- **AHB Slave**

A bus slave responds to a write-read operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.

- **AHB Arbiter**

The bus arbiter have work to ensures that one bus master at a time is granted with bus access and only it is allowed to initiate data transfers. The arbitration protocol is fixed, but we can implement different arbitration algorithm also, such as first come first or priority based arbitration as per the application requirements. An AHB would include only one arbiter, sometimes it is not required in AHB implementation with single bus master.

- **AHB Decoder**

The decoder is used to decode the address for each transfer. It provides a select signal for the slave which is going to involved in the transfer transaction. One centralized decoder is required for AHB implementation.

2.3 AHB Architecture:

The AMBA AHB bus protocol is designed to be used with a central multiplexor interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexor, which selects the appropriate signals from the slave that is involved in the transfer.

AMBA AHB design may contain one or more than one master, a typical system will contain at least the processor and testing the interface. The typical AHB INTERFACE system design includes the following elements:

- Ahb master: By providing the address and control information and the launching of read and write operations. Any time allows only one bus host is in a valid state and can use bus.

- Ahb slave: from the given address space within the scope of the response to the read and write operations. Bus from the machine will be OKAY, ERROR and waiting for data transfer signal back to the current host.

- Ahb bus arbiter: Ensure that every time there was only one bus host is allowed to be bus and initiates the transfer of data. Ahb INTERFACE must contain only a single quorum server, although in a single bus host system. This was not important.

- Ahb address decoder: AHB INTERFACE translator to each time the transfer is the address decoding and transmission contains a selection from the signal. All AHB INTERFACE implementation must only want a central translator.

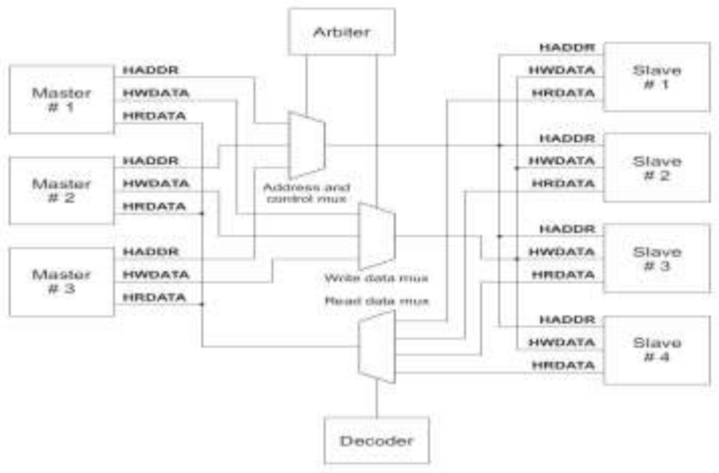


Fig -2 :AHB Nus Interconnection [1]

2.4 Transfer of AHB:

Before start of AMBA AHB transfer the bus master has to be issued access to the bus. Firstly master asserts request signal at arbiter then the process of handovering of bust starts. Then the arbiter has to indicate the master is granted to use the bus.

There are three types of transfers decided by a signal named HBURST, which are fixed, incrementing burst and wrapping burst. After driving the address and control signal by master (driver) the response from slave is like if transfer happens correctly then OKAY response is used, and this response is given by signal HRESP[1:0]. Before first master transfer completes arbiter does not give grant to other master as AMBA AHB allow only one master to use the bus at a time.

2.4.1. Basic Transfer:

The AHB transfer has two distinct phases

1) Address phase, which lasting only for a single cycle.
2) Data phase, may need more than one cycle. This is done by using the HREADY signal.

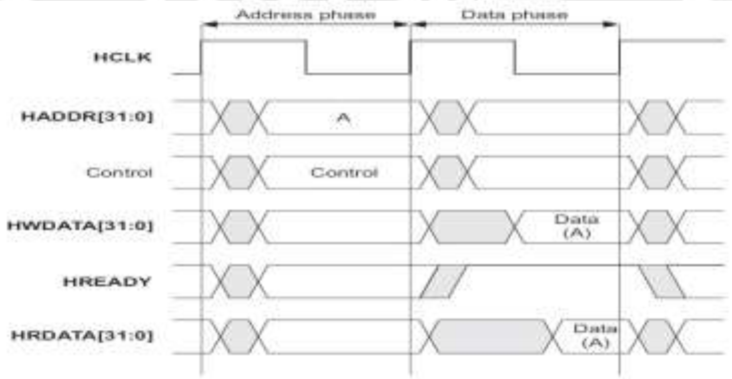


Fig -3 :AHB Basic Transfer [1]

3. VERIFICATION ENVIORNMENT:

Verification process involves the major phases like reviewing, testing, inspecting and documenting the system behavior with respect to the design requirement specifications. The proposed verification environment for my thesis work is Assertion Based Verification. ABV improves the design observability and detects the faults in very fast manner reducing the verification time. Assertion Based Verification is one of the recommended verification techniques to add the quality of verification and helps in debugging time reduction of complex system-on-chip designs. ABV is a method in which to detect specific design behavior, assertions are used either through formal verification, emulation or simulation, of these assertions. In today's scenario, ABV has been well accepted among the design and verification team and it is playing an important role in the verification phase. Assertions detect the malfunctions and thus minimal efforts are required to establish the exact failure reason. Using ABV for AHB design helps to speeds up the process of verification. In verification process, debugging is classified into three stages, at very first instinct detection of error takes place, then in error diagnosis phase root cause to be found and lastly error corrected with help of design reference. SystemVerilog Assertion (SVA) is a type of ABV[2]. SystemVerilog Assertion is a formal specification and verification language. It is also an integral part of SystemVerilog. SVA is a declarative language which has enormous control over time. It is used to describe design properties definitely and precisely. SVA provides several in-built functions for analysis of certain design behavior and also provides temporal domain functional coverage.

3.1 Testbench Architecture:

Thus following environment gives structural description for the AHB verification.

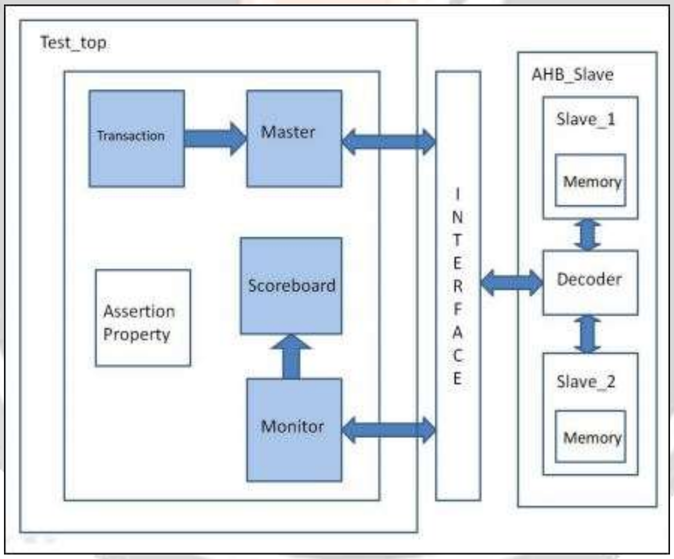


Fig -4: AHB Verification Enviornment

Above environment contain following component. Each component is having specific functionality to be formed as complete Testbench.

- Test_top
- Master
- Ahb_if
- Assertionn
- Monitor
- Scoreboard
- Slave(Ahb_Memory)

We will discuss all components one by one.

1) Test_top: Test_top module contains instances of the all components. It is the top most module in the Testbench hierarchy. This block contains all the test cases implemented.
2) Master: Master is the module in AHB interconnect which initiate the request to access the bus for the transactions to read and write purpose. It is programmed using Bus Functional Model Approach. It contains a host of tasks to do read/write that encompass the way the DUT is used.
3) Ahb_if : Ahb_if stands for the interface in the AHB protocol. Interface contains all the signals required for the AHB. These signals are used to connect the master and slave through the various component. Interface also consists of the global signals like Clock and Reset. Modports do present in the interface. Modports provide the accessibility for the multiple signal with single handle definition.
4) Assertion: In the assertion block all the Boolean function of assertion are written. And with help of property all assertion functionality is evaluated.
5) Monitor This module just monitors the bus traffic on the AHB interface. Monitor checks the protocol is being followed by ongoing transaction using assertion. And the results are given to the scoreboard.
6) Scoreboard: From the data recovered from the monitor scoreboard gives the count for the assertion evaluated, fail count and pass count. Scoreboard gives output on the transcript window of questasim simulator.
7) Slave (Ahb_Memory): Slave is a just memory module, which is also our DUT and it is synthesizable design. This module is programmed to be parameterized to the number of slaves.

3.1 Implementation of Assertions:

Creation of SVA involves multiple stages. Firstly we have to create the Boolean expression in which the functionality is defined in the combination of logical events. These Boolean expression evaluates over the few clock cycles forming the sequences. A more complex sequence is called the property. It includes more number of sequences combined logically or sequentially. We verify the property during a simulation and hence has to be asserted to take effect during a simulation. SVA gives a term to it as "assert" to check the property.

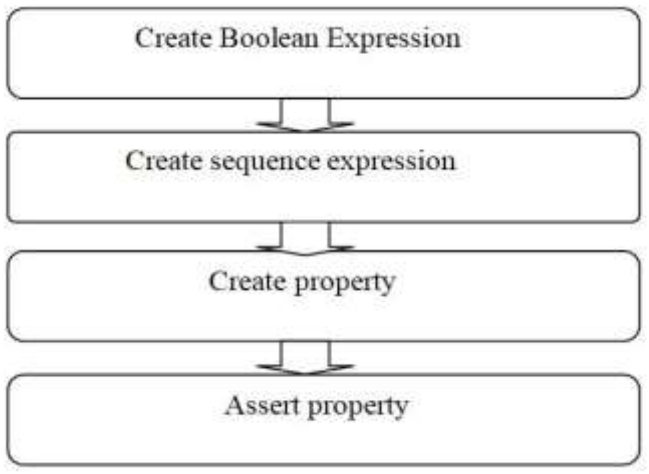
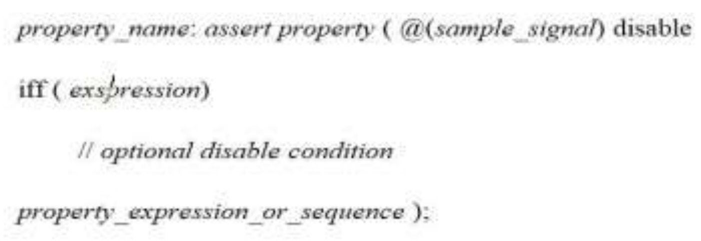


Fig -5: SVA Building block

Following case shows the example for creation of assertion property



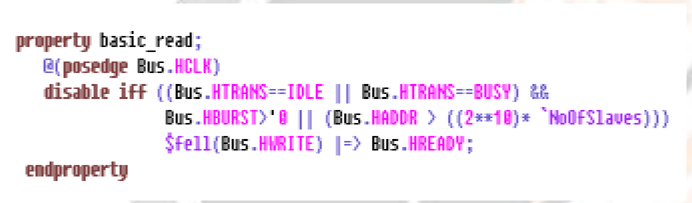
```
property_name: assert property ( @(sample_signal) disable
iff ( exspression)
    // optional disable condition
property_expression_or_sequence );
```

3.2 Property Description of AMBA using SVA properties:

Based on the protocol description of the AMBA-AHB, the following SVA are written in the Testbench. The basic transfer of signals from master to slave and vice-versa are checked by these properties. Creation of such intermediate expressions makes the SVA checks more readable.

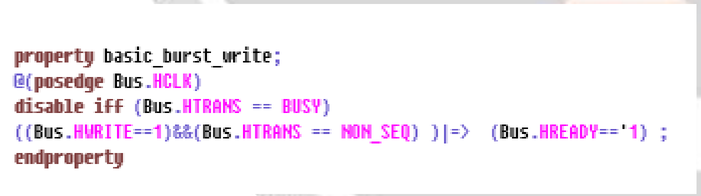
Lets see some of the assertion properties from the project to get understanding about creation and asserting them.

1) Property for basic read operation check.
Assertion defined for basic read, hwrite is detected low at the 2nd clock andthen at the third clock, hready goes high. It is disabled if htrans is either ideal or busy.



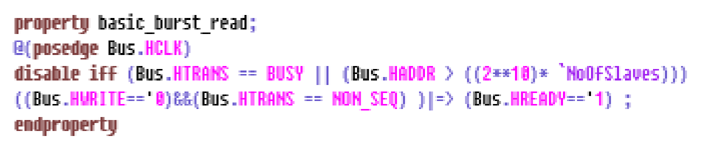
```
 property basic_read;
   @(posedge Bus.HCLK)
   disable iff ((Bus.HTRANS==IDLE || Bus.HTRANS==BUSY) &&
               Bus.HBURST>'0 || (Bus.HADDR > ((2**10)* `NoOFSlaves)))
               $fell(Bus.HWRITE) |=> Bus.HREADY;
  endproperty
```

2) Property for basic burst write operation check.
Assertion defined for burst write, hwrite is detected high & htrans is in the non sequential state at the 2nd clock, at the third clock, hready goes high. It disabled if htrans is busy in state.



```
property basic_burst_write;
@(posedge Bus.HCLK)
disable iff (Bus.HTRANS == BUSY)
((Bus.HWRITE==1)&&(Bus.HTRANS == NON_SEQ) )|=>  (Bus.HREADY=='1) ;
endproperty
```

3) Property for basic burst read operation check.
Assertion defined for burst read, hwrite is detected low & htrans is in non sequential state at the 2nd clock. And at the third clk, hready goes high. Disabled if it is htrans in busy state.

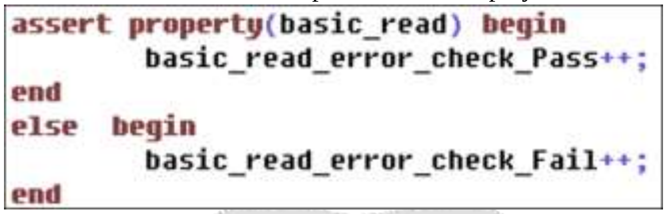


```
property basic_burst_read;
@(posedge Bus.HCLK)
disable iff (Bus.HTRANS == BUSY || (Bus.HADDR > ((2**10)* `NoOFSlaves)))
((Bus.HWRITE=='0)&&(Bus.HTRANS == NON_SEQ) )|=> (Bus.HREADY=='1) ;
endproperty
```

Once these properties are builded they are ready to be asserted in the Testbench architecture. Asserting the properties is similar to if loop used in any basic language constructs. By asserting property we are only checking whether it is failing or passing with the functionality of DUT. If the property passes then pass count is incremented or else fail count has to be incremented. Finally the scoreboard tracks down the number of property asserted, failed and passed properties.

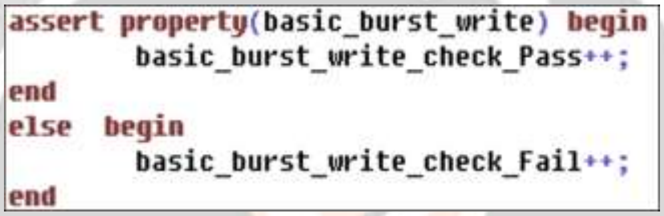
Below mentioned are the properties asserted for checking basic read ,basic burst read , basic burst write operation check.

1) Assert Property for basic read operation check Here we will assert basic read property where if this property pass it will count and increment basic read error check pass and will see the result in scoreboard else if fail it will count and increment basic read error check pass and will display result in scoreboard.



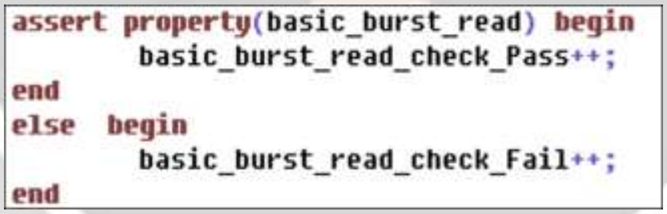
```
assert property(basic_read) begin
        basic_read_error_check_Pass++;
end
else  begin
        basic_read_error_check_Fail++;
end
```

2.) Assert Property for basic burst write operation check Here we will assert basic burst write property where if this property pass it will count and increment basic burst write check pass and will see the result in scoreboard else if fail it will count and increment basic read error check pass and will display result in scoreboard.



```
assert property(basic_burst_write) begin
        basic_burst_write_check_Pass++;
end
else  begin
        basic_burst_write_check_Fail++;
end
```

3) Assert Property for basic burst read operation check Here we will assert basic read property where if this property pass it will count and increment basic burst read check pass and will see the result in scoreboard else if fail it will count and increment basic read error check pass and will display result in scoreboard.



```
assert property(basic_burst_read) begin
        basic_burst_read_check_Pass++;
end
else  begin
        basic_burst_read_check_Fail++;
end
```

4. SIMULATION RESULTS:

Here we are going to discuss the simulation results we have achieved. And also the output of the scoreboard. The simulation is carried out on QuestaSim 10.4e for a data bit of 32 bit size. This platform tool is designed by the Mentor Graphics (now termed as Mentor a Siemens Business). Here the verification analysis is carried out in three cases in system Verilog.

4.1 Case 1 Simulation : 4 Beat Burst Write Read transaction:

We have written test case for 4 Beat Burst Write transaction which can be seen in figure 6 where for the first clk the HADDR is at 32’h0000700 and the data is written at the next clk i.e. HWDATA is 32’h0000001c when only HWRITE goes high i.e. 1’h0 and HREADY goes high i.e. 1’h1 . HBRUST is 3’h3 is increment 4 , the HTRANS is Non-sequential at first clk pulse and sequential from next clk pulse onwards. Response is 1’h0 which represents the response i.e. HRESP as OKAY.

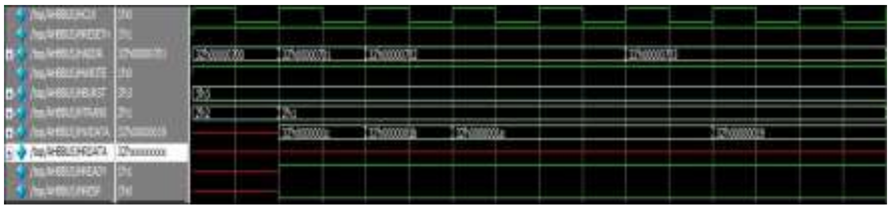


Fig -6: Four Beat Burst Write Transaction

Figure 7 shows 4 Beat Burst Read transaction where for the first clk the HADDR is at 32’h0000700 as the data is written on a particular address, after the data is written on the address the data is to read on the same address, so at HADDR 32’h0000700 the same data is read on same address location i.e. when HREAD is 1’h1 and with response i.e. HRESP as OKAY.

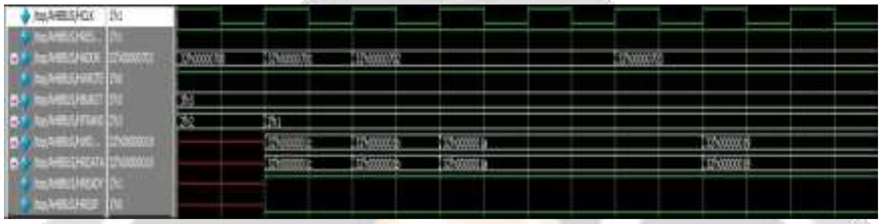


Fig -7: Four Beat Burst Read Transaction

4.2 Case 2 Simulation : 8 Beat Burst Write Read transaction:

Figure 8 shows test case for 8 Beat Burst Write transaction where for the first clk the HADDR is at 32’h0000700 and the data is written at the next clk i.e. HWDATA is 32’h0000001c when only HWRITE goes high i.e. 1’h0 and HREADY goes high i.e. 1’h1 . HBRUST is 3’h5 is increment 8, the HTRANS is Non-sequential at first clk pulse and sequential from next clk pulse onwards. Response is 1’h0 which represents the response i.e. HRESP as OKAY.

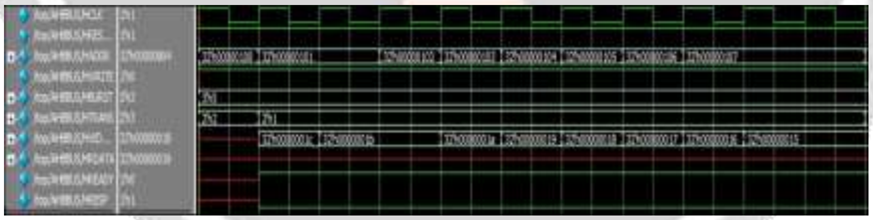


Fig -8: Eight Beat Burst Write Transaction

Figure 9 shows 8 Beat Burst Read transaction for the first clk the HADDR is at 32’h0000700 as the data is written on a particular address, after the data is written on the address the data is to read on the same address, so at HADDR 32’h0000700 the same data is read on same address location i.e. when HREAD is 1’h1 and with response i.e. HRESP as OKAY.

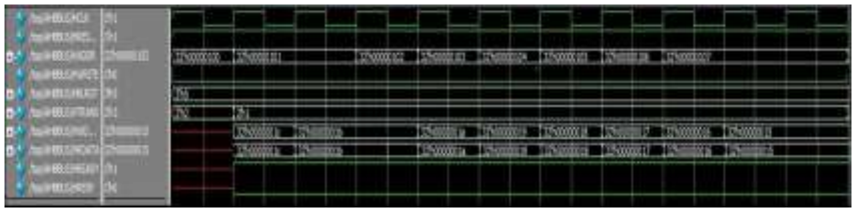


Fig -9: Eight Beat Burst Read Transaction

4.3 Case 3 Simulation : 16 Beat Burst Write Read transaction:

Figure 10 shows test case for 16 Beat Burst Write transaction where for the first clk the HADDR is at 32’h0000700 and the data is written at the next clk i.e. HWDATA is 32’h0000001c when only HWRITE goes high i.e. 1’h0 and HREADY goes high i.e. 1’h1 . HBRUST is 3’h7 is increment 16 , the HTRANS is Non-sequential at first clk pulse and sequential from next clk pulse onwards. Response is 1’h0 which represents the response i.e. HRESP as OKAY.

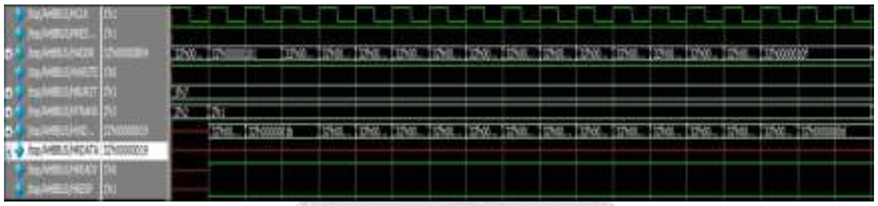


Fig -10: Sixteen Beat Burst Write Transaction

Figure 11 shows 16 Beat Burst Write transaction where for the first clk the HADDR is at 32’h0000700 as the data is written on a particular address, after the data is written on the address the data is to read on the same address, so at HADDR 32’h0000700 the same data is read on same address location i.e. when HREAD is 1’h1 and with response i.e. HRESP as OKAY.

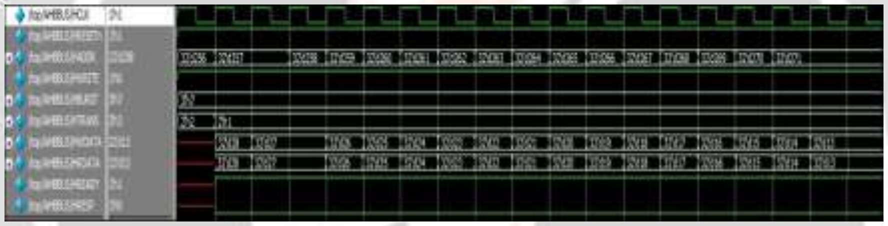
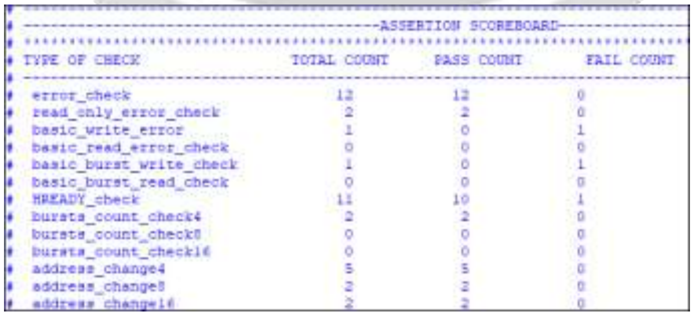


Fig -11: Sixteen Beat Burst Read Transaction

4.4 Scoreboard Output:

For our assertion based verification we have created a scoreboard which tracks the all the information during the simulation time shown in figure 12. Scoreboard is written such that it gives tabular data on the Questa transcript window. It is the complete information about simulation and gives number of properties evaluated on the BFM their pass count and fail count.



```
# ------------------------------------------ASSERTION SCOREBOARD--------------
# ****************************************************************************
# TYPE OF CHECK                TOTAL COUNT      PASS COUNT        FAIL COUNT
# ----------------------------------------------------------------------------
#  error_check                         12              12                 0
#  read_only_error_check                2               2                 0
#  basic_write_error                    1               0                 1
#  basic_read_error_check               0               0                 0
#  basic_burst_write_check              1               0                 1
#  basic_burst_read_check               0               0                 0
#  HREADY_check                        11              10                 1
#  bursts_count_check4                  2               2                 0
#  bursts_count_check8                  0               0                 0
#  bursts_count_check16                 0               0                 0
#  address_change4                      5               5                 0
#  address_change8                      2               2                 0
#  address_change16                     2               2                 0
```

Fig -12: Assertion Scoreboard

5. CONCLUSION:

In this paper the Assertion based verification environment is builded for the AMBA AHB protocol. One Observation is made that assertion verification is faster than the conventional. The idea of this project is to understand and check the working of the protocol.

The proposed verification environment is done in system verilog language construct. The basic feature of AHB like simple read/write, incremental burst read write and multi beat read/write operations are implemented and simulated. And with the help of Scoreboard the depth of functional verification is measured.

All the work and simulation are performed with the help of QuestaSim 10.4e tool (by mentor Graphics).

6. FUTURE SCOPE:

As we know that Amba AHB, this project implementation has to be improved fort the wrapping address boundry of 4k. The ideal AHB protocol is able to serve 64 and 128 bytes of transfer but ut makes more complex design and tidius task to verify. Slave implementation with error and split capability can be introduced.

7. REFRENCES:

[1]ARM, "AMBA Specification(Rev 2.0)",

[2]Srinivas, M. B. R., and Sarada Musala, "Verification of AHB LITE Protocol for Waited Transfer Responses Using Re-Usable Verification Methodology", *International Conference on Inventive Computation Technologies (ICICT)*, 2016.

[3]Gurha, Prince, and R. R. Khandelwal, "SystemVerilog Assertion Based Verification of AMBA-AHB" , *International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, 2016.

[4]Patil, Rohita P., and Pratima V. Sangamkar. A Review of System-On-Chip Bus Protocols. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, vol. 04, no. 01, 2015, pp. 271281., doi:10.15662/ijareeie.2015.0401042.