International Journal of Research in Engineering, Science and Management
Volume-1, Issue-12, December-2018
www.ijresm.com | ISSN (Online): 2581-5792

Verification of Advanced High Speed Bus in UVM Methodology

Malla Siva Ramakrishna[1], Badireddy Satya Sridevi[2]
[1]*Student, Dept. of Electronics and Communications Engg., Aditya Engineering College, Peddapuram, India*
[2]*Professor & HoD, Dept. of Electronics and Communications Engg., Aditya Engg. College, Peddapuram, India*

***Abstract*: This paper describes the verification of AHB Protocol using the methodology UVM (Universal Verification Methodology). AHB is an Advanced High performance system Bus that supports multiple masters and multiple slaves. It implements burst transfers, split transactions, single-cycle bus master handover, single-clock edge operation, wider data bus configuration (64/128bits). Verification IP is the one which provides a smart way to verify the AHB Components such as Master, Slave, Arbiter and Decoder. UVM is used for the verification of AHB Protocol which provides the best framework to achieve CDV (Coverage Driven Verification) which combines automatic test generation, self-checking test benches, and coverage metrics to significantly reduce the time spent on verifying a design. An UVM test bench is composed of reusable verification environments called VCs (verification Components). This paper examines the verification of VCs which are structured to work with Verilog, System Verilog and System C.**

***Keywords*: AHB, CDV, UVM, VCs, TB, Sequencer**

1. Introduction

AMBA–AHB is intended to address the high performance synthesizable design. AHB is the new level bus implements for the high performance applications and high clock frequency systems. UVM is a complete verification methodology which extends from System Verilog and OVM .Which is targeted at verifying large gate count and IP based SOC. Verification productivity stems from the ability to quickly develop individual verification components, encapsulate them into larger reusable verification components (VCs), and reuse them in different configurations and at different levels of abstraction. UVM uses a System Verilog implementation of standard Transaction Level Modelling (TLM) interfaces for modular communication between AHB components such as Master and Slave. The System Verilog UVM Class Library also provides various utilities to simplify the development and use of verification environments. These utilities support debugging by providing a user- controllable messaging utility. The System Verilog UVM Class Library provides global messaging facilities that can be used for failure reporting and general reporting purposes.

A. Objective of study

1) To study the specifications of AMBA AHB which include all the scenarios.
2) To generate the Test Plan comprises of Test Cases which meets the specified scenarios.
3) To understand the development of Verification Environment for Single Master -Single Slave which follows the UVM topology of Driver, Sequencer and a Monitor along with Test Plan
4) Analysis on how the driver drives the sequences from the sequencer to the Score Board.
5) In the scoreboard, the actual output is compared with the expected one. If the obtained output matches with the expected result means verification is completed successfully.

2. Architecture of UVM test bench

A. UVM test bench and environment

An UVM test bench is composed of reusable verification environments called Verification Components (VCs). The

VCs are applied to the device under test (DUT) to verify the implementation of the AHB protocol. Architecture of UVM n test bench is shown in Fig. 1.

B. Building blocks of test bench

The three main building blocks of a test bench in UVM based verification are,

- Uvm_env:

It is the top level component of the verification components. uvm_env is extended from uvm_component. It is used to create and connect the uvm_components like drivers, monitors, sequencers. It can also use as sub environment in another environment.

- UVM test:

The uvm_test class defines the test scenario for the test bench specified in the test. The test class enables configuration of the test bench and verification components.

- UVM Verification Components:

Sequencer (stimulus generator): The sequencer generates stimulus data and passes it to a driver for execution. The uvm_sequencer is the base class of uvm class library contains all of the base functionality required to allow a sequence to communicate with a driver.

1. Driver: The driver drives data items to the bus following the interface protocol. The driver obtains the data items from the sequencer for execution. The UVM Class Library provides the uvm_driver base class.
2. Monitor: The monitor extracts signal information from the bus and translates it into events, structs, and status information.
3. Agent: An agent has two basic operating modes

i. Active mode: In this mode, the agent emulates a device in the system and drives DUT signals. This mode requires that the agent instantiate a driver and sequencer. A monitor also is instantiated for checking and coverage.

ii. Passive mode: In this mode, the agent does not instantiate a driver or sequencer and operates passively. Only the monitor is instantiated and configured. This mode is used only when checking and coverage collection is desired.

4. UVM top:

Test Bench top is the module, it connects the DUT and Verification environment components.

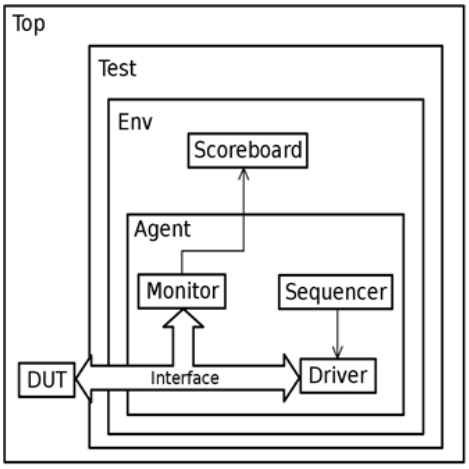

Fig. 1. Typical UVM test bench architecture

3. UVM class hierarchy

The components in an UVM verification environment are derived either directly or indirectly from the uvm_component class as shown in Fig. 2.

- Build(): This phase is used to construct various child components/ports/exports and configures them.
- Connect(): This phase is used for connecting the ports/exports of the components.
- End of elaboration(): This phase is used for con gusing the components if required.
- Start of simulation(): This phase is used to print the banners and topology.
- Run(): In this phase, main body of the test is executed where all threads are forked off.

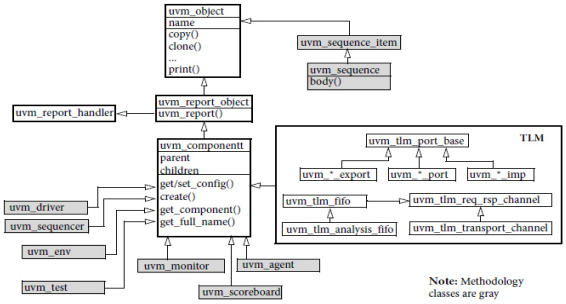

Fig. 2. UVM class hierarchy

4. AMBA AHB features

A. *Design methodology*

This includes the features of AMBA AHB protocol and its Components like master, slave, arbiter, decoder and its interconnections.

B. *AMBA AHB protocol features*

AMBA AHB implements the features required for high performance, high clock frequency systems including:

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

Bridging between this higher level of bus and the current ASB/APB can be done efficiently to ensure that any existing designs can be easily integrated. An AMBA AHB design may contain one or more bus masters, typically a system would contain at least the processor and test interface. However, it would also be common for a Direct Memory Access(DMA) or Digital Signal Processor(DSP) to be included as bus masters. The external memory interface, APB bridge and any internal memory are the most common AHB slaves. Any other peripheral in the system could also be included as an AHB slave.

C. *A typical AMBA AHB system contains the following components:*

1) *AHB bus interconnection*

The AMBA AHB bus protocol is designed to be used with a Central multiplexer interconnection scheme. A central decoder is also required to control the read data and response signal multiplexer, which selects the appropriate signals from the slave that is involved in the transfer.

2) *AHB master interface diagram*

The interface diagram of an AHB bus master shows the main signal groups. An AHB master provides address and control information to initiate read and write operations.

3) *AHB Slave Interface diagram*

The interface diagram of an AHB bus slave shows the main

signal groups. An AHB slave responds to transfers initiated by masters in the system. The slave uses the HSELx select signal from the decoder to control when it responds to a bus transfer.

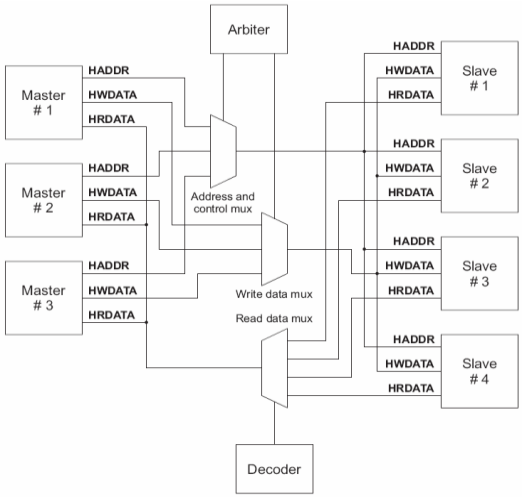

Fig. 3. Multiplexer interconnection

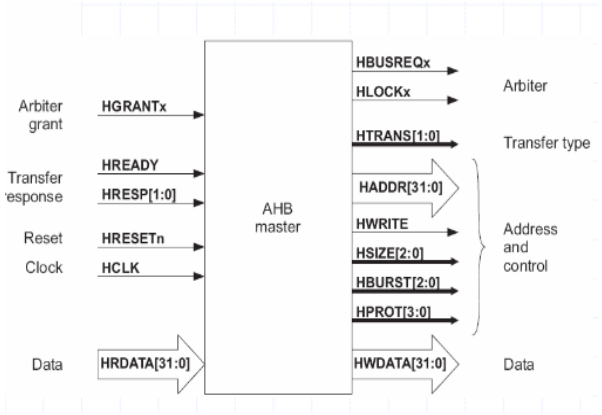

Fig. 4. AHB bus master interface diagram

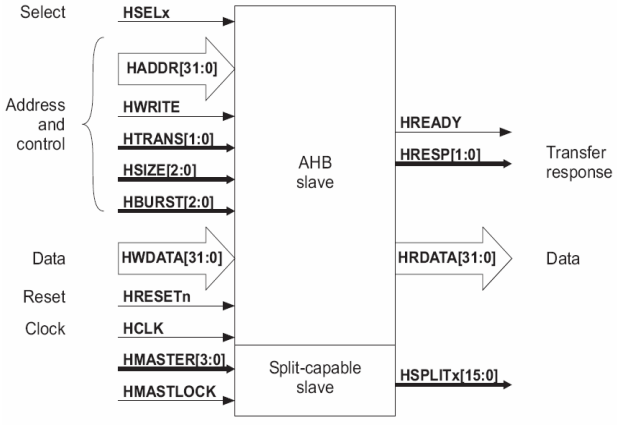

Fig. 5. AHB bus slave interface diagram

4) AHB decoder

This component decodes the address of each transfer and provides a select signal for the slave that is involved in the transfer. It also provides a control signal to the multiplexor. A single centralized decoder is required in all AHB-Lite implementations that use two or more slaves.

In multi-layer AHB-Lite implementations, the decoder function is usually included in the multi-layer interconnect component.

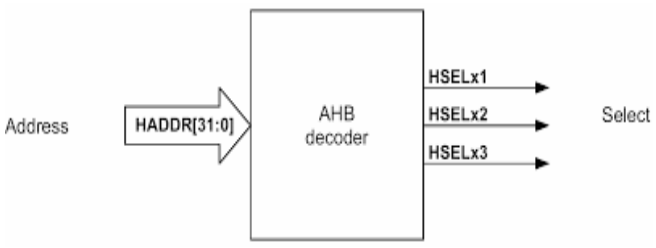

Fig. 6. AHB decoder diagram

5) AHB arbiter

The role of an arbiter in an AMBA system is to control which master has access to the bus. Every bus master has a REQUEST/GRANT interface to the arbiter and the arbiter uses a prioritization scheme to decide which bus master is currently the highest priority master requesting the bus. Each master also generates an HCLOCKx signal which is used to indicate that the master requires exclusive access to the bus. The detail of the priority scheme is not specified and defined for each application. It is acceptable for the arbiter to use other signals, either AMBA or non-AMBA, to influence the priority scheme that is in use.

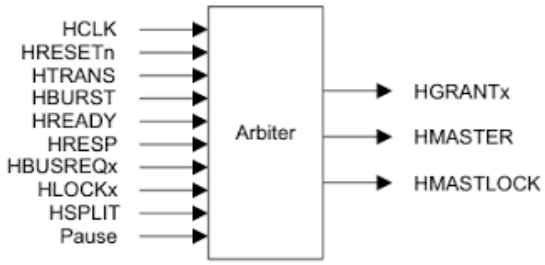

Fig. 7. AHB arbiter diagram

5. Results and discussion

The results of verification components such as Master Agent and the Slave Agent of the UVM Environment are presented. According to Test Plan, the test cases are verified by developing the Verification IP for the AHB Protocol. The Test Cases are written in the form of sequences in the Sequencer using System Verilog. The sequencer drives the sequences to the driver and thereby to Score Board. In the scoreboard, the actual output is compared with the expected one. If the obtained output matches with the expected result then we conclude that the verification is completed successfully. By using the tool Questa, the Verification of AHB Components known as VC's such as Master Agent and Slave agent are done and the log files for the test cases are generated and simulated waveforms are achieved and shown in Fig. 8 to Fig. 11.

International Journal of Research in Engineering, Science and Management
Volume-1, Issue-12, December-2018
www.ijresm.com | ISSN (Online): 2581-5792

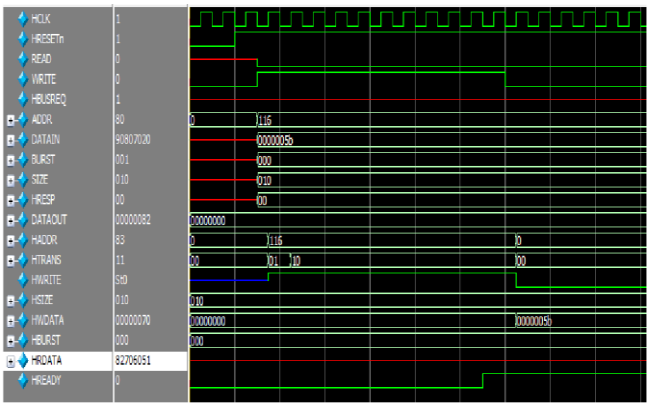

Fig. 8. AHB single burst

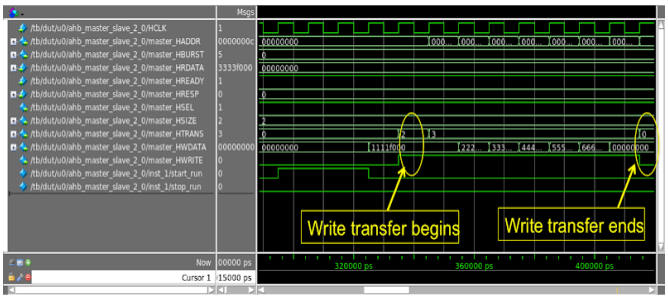

Fig. 9. AHB write transfer

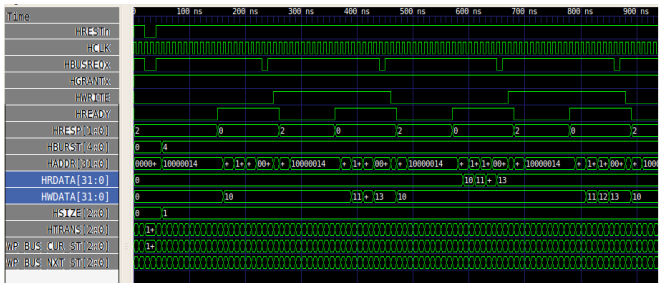

Fig. 10. AHB write and read data

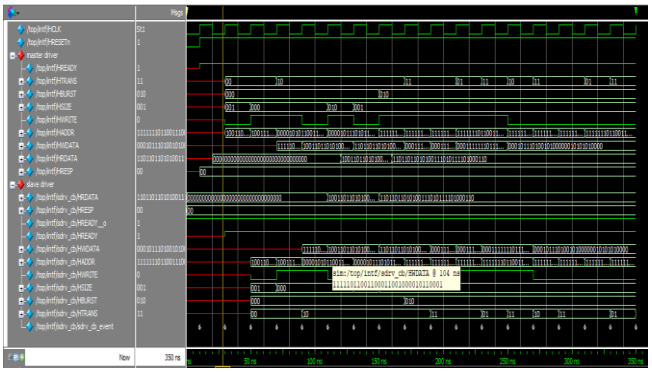

Fig. 11. Master and slave data transfer

A. *UVM Info*

```
# ----------------------------------------------------------------
# UVM-1.0p1
# (C) 2007-2011 Mentor Graphics Corporation
# (C) 2007-2011 Cadence Design Systems, Inc.
# (C) 2006-2011 Synopsys, Inc.
# ----------------------------------------------------------------
# UVM_INFO @ 0: reporter [RNTST] Running test reset_test...
# ----------------------------------------------------------------
# Name                        Type                      Size  Value
# ----------------------------------------------------------------
# uvm_test_top                reset_test                -     @463
#   env_h                     ahb_env                   -     @478
#     ahb_coverage_h          ahb_coverage              -     @553
#       analysis_imp          uvm_analysis_imp          -     @560
#     ahb_master_ap           uvm_analysis_port         -     @485
#     ahb_slave_ap            uvm_analysis_port         -     @493
#     master_agent_h          ahb_magent                -     @523
#       agent_ap              uvm_analysis_port         -     @530
#       mdriver_h             ahb_mdriver               -     @701
#         rsp_port            uvm_analysis_port         -     @716
#         sqr_pull_port       uvm_seq_item_pull_port    -     @708
#       mmonitor_h            ahb_mmonitor              -     @686
#         monitor_ap          uvm_analysis_port         -     @693
#       mseqr_h               ahb_mseqr                 -     @724
#         rsp_export          uvm_analysis_export       -     @731
#         seq_item_export     uvm_seq_item_pull_imp     -     @825
#         arbitration_queue   array                     0     -
#         lock_queue          array                     0     -
#         num_last_reqs       integral                  32    'd1
#         num_last_rsps       integral                  32    'd1
#     reset_agent_h           reset_agent               -     @516
#       driver_h              reset_driver              -     @845
```

B. *UVM report*

```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   29
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [RNTST]     1
# [TEST_DONE]     1
# [ahb_coverage]     1
# [ahb_mdriver]     8
# [ahb_mmonitor]    10
# [ahb_smonitor]     8
```

6. Conclusion

This paper presented verification of advanced high speed bus in UVM methodology.

References

[1] Jack Erickson, “TLM-Driven Design and Verification – Time for a Methodology Shift”, Cadence Design Systems, Inc.
[2] Rath A.W, Esen.V and Ecker.W , “A transaction –oriented UVM-based library for verification of analog behavior ” Publication Year:2014 Page(s): 806 – 811.
[3] Stuart Sutherland, Don Mills, “Synthesizing System Verilog Busting the myth that System Verilog is only for Verification”
[4] Soo-Yun Hwang and Kyoung-Sun Jhang , “An Improved Implementation Method Of AHB BusMatrix” SOC Conference 2005, Proceedings, IEEE International pp. 211-214.
[5] Mulani, “ Level Verification Using SystemVerilog ” Emerging Trends in Engineering and Technology (ICETET), 2009 2nd International Conference on 16-18 Dec.2009 pp. 378-380.

[6] Pockrandt, M , Herber, P and Glesner, S, " Model checking a SystemC/TLM design of the AMBA AHB Protocol " Embedded Systems for Real-Time Multimedia (ESTIMedia), 2011 9th IEEE Symposium on 13-14 Oct.2011 pp.66 – 75
[7] IEEE Draft Standard for System Verilog - Unified Hardware Design, Specification and Verification Language, IEEE P1800/DS, February 2012 pp.1-1304.
[8] Young-Nam Yun, Jae-Beom Kim, Nam-Do Kim and Byeong Min, "Beyond UVM for practical SoC verification" SoC Design Conference (ISOCC), 2011 International pp. 158-162
[9] Keaveney. Martin, Mc Mahon. Antony, O' Keeffe. Niall, Keane. Kevin and O.Reilly James "The Development of advanced verification environments using System Verilog" Signals and Systems Conference,208.(ISSC 2008),IET Irish pp.325-330
[10] Universal Verification Methodology 1.1 User's Guide May 18, 2011. http://www.accellera.org