ECE-593 Fundamentals of Pre-Silicon Validation
Final Project Report
Spring 2021

UVM-Environment based verification of AHB-Lite protocol

By: Sri Harsha Doppalapudi, Shivanand Reddy Gujjula, Hiranmaye Sarpana Chandu

Under the Esteemed Guidance of
DR. TOM SCHUBERT BA, Ph.D.,
Director of ECE Design Verification and Validation
Graduate Program Track
Portland State University

Table of Contents

1. Protocol Description

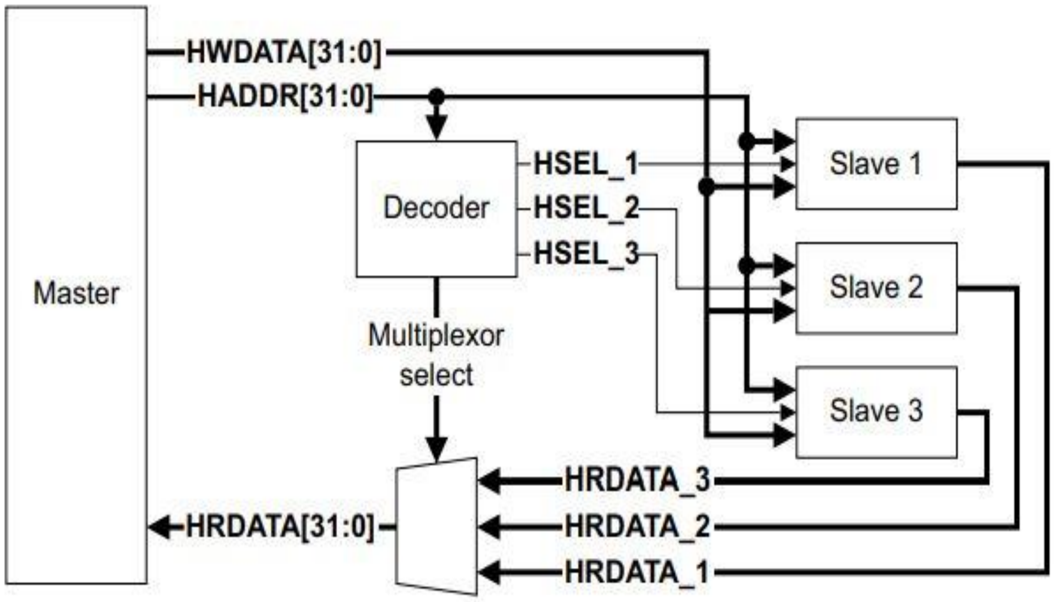
1.1 About the Protocol

AMBA AHB-Lite addresses the requirements of high-performance synthesizable designs. It is a bus interface that supports a single bus master and provides high-bandwidth operation. The most common AHB-Lite slaves are internal memory devices, external memory interfaces, and high bandwidth peripherals.

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

- Burst Transfers
- Single-clock edge operation
- Non-tristate implementation
- Wide data bus configurations, 64, 128, 256, 512, and 1024 bits (in this design is implemented only for 32 bit)

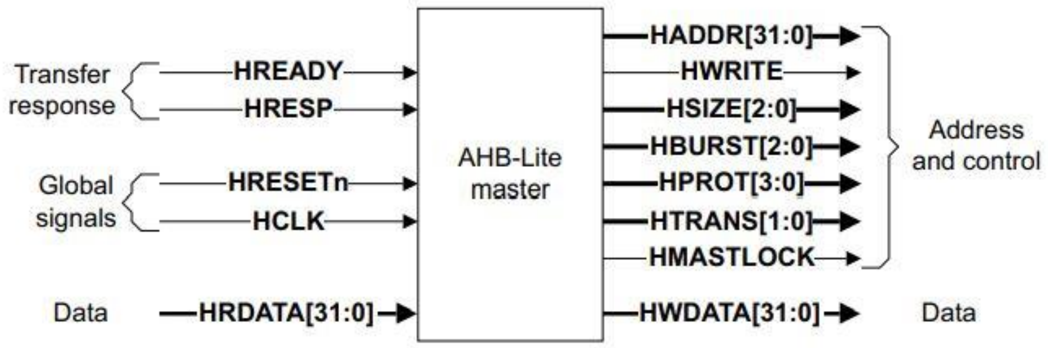
Below figure demonstrates a single master AHB-Lite design with one AHB-Lite master and three AHB-Lite slaves. The bus interconnect logic consists of one address decoder and a slave-to-master multiplexor. The decoder monitors the address from the master so that the appropriate slave is selected and the multiplexor routes the corresponding slave output data back to the master.



1.2 Components of protocol

1.2.1 Master

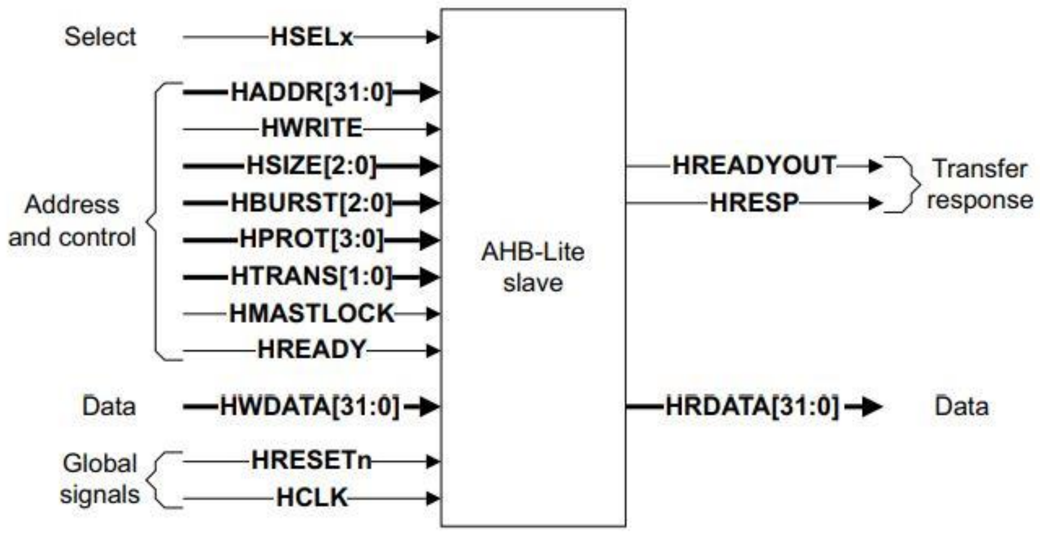
Master provides address and control information required by slave to initiate read and write transfers.



1.2.2 Slave

AHB_Lite slave responds to transfers initiated by master in the system. Slave responds only when HSELx signal of respective slave is asserted. Slave responds with three different signals:

- HRESP as OKAY representing successful transfer
- HRESP as ERROR representing failure in transfer
- HREADY is de-asserted representing a wait state to master(not implemented as part of this design taken for this project)



1.2.3 Decoder

This component decodes the address of each transfer and provides HSELx signals for all the slaves in the system with only one of them asserted. It also provides control signal for the multiplexor.

1.2.4 Multiplexor

This component uses the control signal sent by decoder and multiplex the read data bus and response signals from the slaves to master.

1.3 Operations

The master starts a transfer by driving the control signals. All the information about address, direction, width of the transfer is provided by these signals. It Provides information about the transfer type like single transfer or multiple transfers depending upon burst. Transfers can be classified as below:

- Single transfers
- Increment bursts
- Wrap bursts that wrap at particular address boundaries

Every transfers above consists of two phases:
Address Phase: This phase is of one cycle and provides address and control signals
Data Phase: This phase can be of one or more cycles, read data from slave and write data from master are provided in this phase.

1.4 Signal Descriptions

1.4.1 Global signals

Name	Source	Description
HCLK	Clock Source	All signal timings are related to posedge of HCLK
HRESETn	Reset Controller	Active Low signal which resets the system and bus

1.4.2 Slave signals

Name	Destination	Description
HRDATA[31:0]	Multiplexor	provides data on the bus during read transfer
HREADYOUT	Multiplexor	Indicates whether transfer is completed on bus
HRESP	Multiplexor	indicates the success and failure of current transfer

1.4.3 Master signals

Name	Destination	Description
HADDR[31:0]	Slave and Decoder	Address bus
HBURST[2:0]	Slave	Indicates the burst type(increment/wrap) and number of bursts(4,8,16,undefined length) in transfer
HMASTLOCK	Slave	specifies whether current transfer is part of locked sequence or not.
HPORT[3:0]	Slave	Provides some additional information about the current transfer like bufferable/cacheable
HSIZE[2:0]	Slave	Indicates size of transfer(8,16,32..etc.,)
HTRANS[1:0]	Slave	Indicates type of trnasfer(IDLE, BUSY, NONSEQ, SEQ)
HWDATA[31:0]	slave	provides data to be written in memory during write transfer
HWRITE	slave	indicates whether it is read or write transfer.

1.4.4 Decoder signals

Name	Destination	Description
HSELx	Slave	Indicates which slave to respond.

1.4.5 Multiplexor signals

Name	Destination	Description
HRDATA[31:0]	Master	Multiplexed output of Read data bus
HREADY	Master and Slave	Indicates whether previous transfer is complete or not.
HRESP	Master	Multiplexed output of HRESP from all slaves

2. Verification Plan

2.1 Introduction

AMBA AHB-Lite is a bus interface that supports a single bus master and provides high bandwidth operation. The most common AHB-Lite slaves are internal memory devices, external memory interfaces, and high bandwidth peripherals.

Source of design: Design source code taken from github **Click here**

Above design implements the slave behaviour of the AHB-Lite protocol with two slaves. Two slaves are memory devices with both having the same memory controller.

Design consists of a decoder which decodes the address in the address phase of each transfer and assert HSELx signal of respective slave. When HSELx of any slave is asserted respective slave responds as per the control signals by placing HRDATA and HRESP on their respective buses. Here a multiplexor with control signal taken from decoder multiplexes the buses and provides HRDATA and HRESP of selected slave to master.

Design implements various transfer types (IDLE, BUSY, SEQ, NONSEQ), various bursts types (SINGLE, INCR, INCR4, WRAP4, INCR8, WRAP8, INCR16, WRAP16) with a fixed transfer size(HSIZE = WORD). It also supports both read and write transfers.

On any transfer master provides the address and control signals in one address phase and write data in data phase of write transfer whereas during read transfer slave places read data on bus during data phase of read transfer. The protocol works in a pipelined nature as the data phase of current transfer is the address phase of next transfer.

2.2 Description of verification levels

As the design consists of only two modules with some simpler components like decoder and multiplexor. Controllability and observability of the inputs and outputs at system level and unit level are almost similar. So, the design will be verified only at the top level.

2.3 Tools Required

Simulation Tool: Questasim
Version Control Tool: Github
Editor Tool: Notepad++
Documentation Tool: MS Office

2.4 Risks and Dependencies

Due to time constraint few corner cases may remain unverified but the goal is to verify all corner cases as well.

2.5 Functions to be verified

2.5.1 Basic/Critical

- Write data bus operation from Master to Slave(both)
- Read data bus operation from Slave(both) to Master
- Single-clock edge operation
- Burst transfers
    - Single burst
    - Undefined length incrementing Burst
    - Incrementing Bursts
    - Wrapping Bursts

2.5.2 Advanced

- Continuous Writes to same address
- Continuous Reads to different addresses
- Continuous Writes to different addresses
- Read-Write to same address
- Write-Read to same address
- Read-Write to different addresses
- Write-Read to different addresses
- Write-Write-Read to same address
- Write-Read-Write to same address
- Read-Write-Read to same address
- Read-Write-Write to same address
- All the above test cases for all BURST types

2.5.3 Generic/Universal

- Reset operation

2.5.4 What Will not be verified

- Locked transfer
- Protected transfer

2.6 Tests and Test Methods

2.6.1 Type of Verification

As the specifications describe only about top-level interfaces, no knowledge of design implementation is required. So, black box verification is suitable for this project.

2.6.2 Verification Strategy

Our test environment supports constrained random stimulus generation, so both random and deterministic stimulus are used.

- **Constrained Randomization Tests**

  The values of address, data, burst size and type are generated within the bus range. As part of the randomization, we would explicitly disable the illegal stimulus. Post-randomization and static values are used to make sure of current randomized stimulus is dependent on previous stimulus (for address generation). As required some test cases will be generated using inline constraints, soft constraints.

- **Deterministic Tests**
  For the simplicity of stimulus generation and to get better coverage according to the plan, individual sequences for different burst types are generated.

- **Abstraction Level**
  Considering the no. of possible input combinations and tests, transaction level abstraction would be appropriate for this project. These transaction-based stimuli (packets with all address, control and data signals required for the transfer) are converted to pin level and driven via interfaces by the driver and output signals are captured and converted from pin level to transaction level by the monitor. The transactions generated are sent to one component to another via TLM ports.

- **Checking**
  The pin level input signals driven, and the output signals are captured and converted into transaction packets by the monitor and sent to scoreboard which consists of a output predictor and output comparison functions.

2.7 Test Environment

The entire verification testbench will be developed using system Verilog classes by adopting Universal Verification Methodology (UVM). Test Environment consists of various components like tests, sequence item, sequences, sequences, virtual sequence, sequencer, virtual sequencer, environment, agent, monitor, driver, coverage and scoreboard. In detail description of each components will be discussed in later sections.

2.8 Coverage Requirements

Coverage goals for the AHB-Lite Protocol are based on the tests defined in the functions to be verified section. Both structural (code) coverage and functional coverage will be collected and analysed. The following coverage options will be used to ensure the functional coverage.

Cover points for all inputs:

- HWRITE
  - Read
  - Write
- HBURST
  - SINGLE
  - INCR
  - WRAP4
  - INCR4
  - WRAP8
  - INCR8
  - WRAP16

    - INCR16
- HTRANS
    - NONSEQ
    - SEQ
    - BUSY
    - IDLE
- HSIZE
    - WORD
- HADDR[10] (selecting both slaves)
- HADDR[9:0]
- HWDATA
- Cross of HTRANS, HBURST, HSIZE, HWRITE, HADDR[10] (while implementing it is written as cross of cross cover points)

Cover points for all outputs:

- HRESP
    - OKAY
    - ERROR
- HRDATA

Cover points for different sequences:

- Write-Read same address
- Read-Write same address
- Write-Write same address

- Write-Read different address
- Read-Write different address
- Write-Write different address
- Read-Read different address

- Write-Write-Read same address
- Write-Read-Write same address
- Read-Write-Read same address
- Read-Write-Write same address

- Cross of different sequences and HTRANS
- Cross of different sequences and HBURST
- Cross of different sequences and HSIZE
- Cross of different sequences and HADDR[10]

2.9 Test case Matrix

Simple operations:

- Write – SINGLE burst
- Read – SINGLE burst
- Write – INCR burst
- Read – INCR burst

- Write – WRAP4 burst
- Read – WRAP4 burst
- Write – INCR4 burst
- Read – INCR4 burst
- Write – WRAP8 burst
- Read – WRAP8 burst
- Write – INCR8 burst
- Read – INCR8 burst
- Write – WRAP16 burst
- Read – WRAP16 burst
- Write – INCR16 burst
- Read – WRAP16 burst
- All the above cases for both the slaves

Back-to-Back operations:

- Write-Read same address
- Read-Write same address
- Write-Write same address
- Write-Read different addresses
- Read-Write different addresses
- Write-Write different addresses
- Read-Read different addresses
- Write-Write-Read same address
- Write-Read-Write same address
- Read-Write-Read same address
- Read-Write-Write same address
- All the above cases with different burst types and for all the slaves.

2.10 Resources and Responsibilities

Sri Harsha Doppalapudi:
Responsible for developing,

- AHB_monitor
- AHB_coverage
- AHB_scoreboard

Shivanand Reddy Gujjula:
Responsible for developing,

- AHB_pkg,
- AHB_sequence_item
- AHB_sequences
- AHB_virtual_sequence
- AHB_sequencer
- AHB_virtual_sequencer
- AHB_driver
- AHB_agent

Hiranmaye Sarpana Chandu:
Responsible for developing,

- AHB_packet
- AHB_environment
- AHB_test
- AHB_TBtop
- AHB_interface

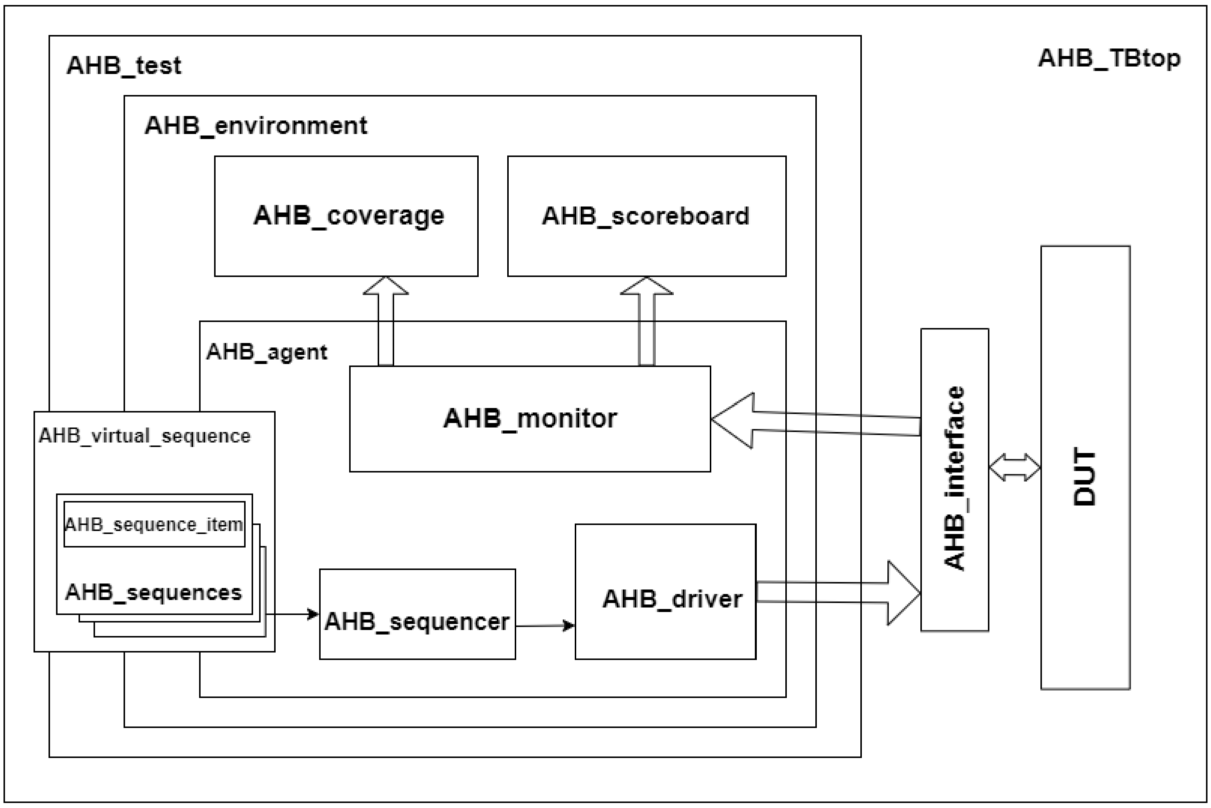
2.11 Schedule Details

Week1 (5/05/2021-11/05/2021)	Analysis and understanding of AMBA AHB-Lite Protocol from the AMBA AHB-LITE Specification document and preparation of verification plan.
Week2 (12/05/2021-18/05/2021)	UVM based object-oriented verification environment development. Debug the test bench bugs. Deterministic Tests run and waveform generation.
Week3 (19/05/2021-25/05/2021)	Top level/System level verification followed by bug fixes. Includes Constrained random tests , coverage collection, self – checking with scoreboard. Identify if any missing corner cases / uncovered scenarios.
Week4 (26/05/2021-01/06/2021)	Documentation and logging of results and observations.

3. Description of UVM Test Environment

Introduction to UVM

The testbench environment is developed using the System Verilog classes and the framework is adopted from the Universal Verification Methodology. The following diagram shows the testbench components and the transaction level communication among the components.



UVM based Testbench Block Diagram

3.1 AHB_TBtop

AHB_TBtop is the top module in the Testbench hierarchy.
Tasks done in AHB_TBtop are:

- Generate clock
- Create AHB_interface handle
- Instantiate the DUT and pass the interface handle
- Set the configuration for the virtual interface using config_db
- Run the test – AHB_test

3.2 AHB_test

Tasks done in AHB_test are:

- Create the AHB_env, AHB_virtual_sequence in the build phase
- Run or start the AHB_virtual_sequence in the run phase

3.3 AHB_sequence_item

AHB_sequence_item is the transaction class which deals with the stimulus generation depending on the constraints defined as per the DUT specification.
The following constraints have been written for this project specification,

- HTRANS
  - HTRANS_SINGLE – For single burst type
  - HTRANS_OTHER – For remaining burst types
- HADDR
  - AddrHighbits – For address bits [31:11]
  - AddrMin – For Min address size
  - Addr – For no. of addresses to be generated based on Burst type
  - ADDR_ALIGNMENT – Based on HSIZE
  - ADDR_INCR – For increment bursts
  - ADDR_WRAP – For wrapping bursts
- HSIZE
  - Hsize – 32 bits
- BUSY STATE
  - BUSY_SIZE – Based on burst type and number of busy states
  - BUSY_POSITION – To indicate when to insert BUSY in the transfer
  - BUSY_COUNT – Number of BUSY states
  - NOBUSY_FIRST – No BUSY during initiating new transfer
  - post_randomize() function is used to insert these BUSY states in the HTRANS[ ] array .

Note: Please refer to the file "AHB_sequence_item.sv" for better understanding

3.4 AHB_sequences

AHB_sequences class creates the transaction object (AHB_sequence_item), randomizes it specific to the burst type, read or write for both slave0 and slave1.Each Burst type has a sequence class which is derived from AHB_base_sequence. AHB_base_sequence consists of,

- hwrite [ ] – which represents the read – write operation pattern
- ADDRESS [ ] – which represents the sequence of addresses for read – write pattern

The above two dynamic arrays are used to develop deterministic tests for Read – Write operations to same and different addresses.

And individual sequence classes are derived from AHB_base_sequence to develop deterministic stimulus for each burst type. The following are the sequences for burst types,

- sequence_SINGLE_burst
- sequence_INCR_burst
- sequence_INCR4_burst
- sequence_INCR8_burst
- sequence_INCR16_burst
- sequence_WRAP4_burst
- sequence_WRAP8_burst
- sequence_WRAP16_burst

Note: Please refer to the file "AHB_sequences.sv" for better understanding.

3.5 AHB_virtual_sequence

AHB_virtual_sequence is the top-level sequence which is initiated in the AHB_test class. It is used to initiate multiple sequences sequentially. It doesn’t need any transaction object.

Tasks done in AHB_virtual_sequence:

- Create all the sequences defined for each burst type in pre_body().
- Run or start the sequences sequentially on their respective sequencers in the body() task.

3.6 AHB_environment

Tasks done in AHB_environment:

- Create the AHB_agent, AHB_scoreboard, AHB_coverage(uvm_subscriber), AHB_virtual_sequencer in build phase
- In connect phase, connect the TLM analysis ports between
    - AHB_monitor in AHB_agent and AHB_scoreboard
    - AHB_monitor in AHB_agent and AHB_coverage
- Assign the AHB_sequencer in AHB_agent to the AHB_sequencer handle in AHB_virtual_sequencer.

3.6.1 AHB_agent

AHB_agent consists of components responsible for generating stimulus, driving signals to DUT and capturing interface signals related to a particular DUT interface.

Tasks done in AHB_agent:

- Create the components AHB_sequencer, AHB_driver and AHB_monitor in build phase.
- In connect phase, connect TLM ports between AHB_driver and AHB_sequencer.

    - **AHB_sequencer:**
      AHB_sequencer is a custom sequencer derived from uvm_sequencer to execute the AHB_sequences.

    - **AHB_driver:**
      AHB_driver is responsible for driving signals to DUT via virtual interface. Tasks done in AHB_driver:
        - Get the virtual interface handle suing config_db set in AHB_TBtop in build phase
        - Wait for the reset phase to complete
        - In run phase,
            - Get the transaction object using get_next_item()
            - Drive the control signals , address and HWDATA to the DUT depending upon HREADY from DUT.
            - After driving the transaction object to DUT, acknowledge sequencer using item_done() indicating present transaction is completed and ready to accept next item and repeat until all items have been driven to DUT.

- **AHB_monitor:**
  AHB_monitor snoops on the interface signals and captures those signals to broadcast to other components. Tasks done in AHB_monitor:
  - Capture the interface signals
  - Convert the pin level signals to transaction objects
  - Broadcast the transaction objects to AHB_coverage and AHB_scoreboard via analysis port

3.6.2 AHB_virtual_sequencer

AHB_virtual_sequencer is a custom sequencer derived from uvm_sequencer to execute the AHB_virtual_sequence. It doesn't need any transaction object.

3.6.3 AHB_scoreboard

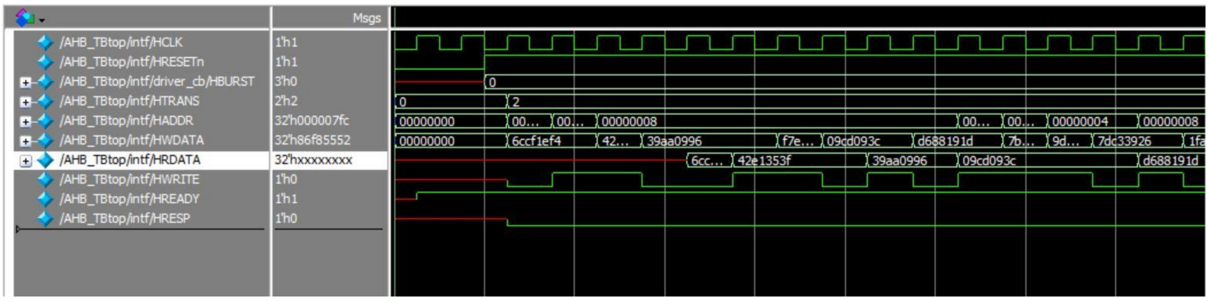
AHB_scoreboard is a class extended from uvm_scoreboard which receives packet of data from AHB_monitor via analysis port when write() function is called in AHB_monitor. On receiving packet, it is placed into a TLM FIFO. During the run phase each packet is taken and outputs of next cycle are predicted. Followed by comparison of predicted outputs with actual outputs. In report phase statistics are reported.

3.6.4 AHB_coverage

AHB_coverage is a class extended from uvm_subscriber which receives packet of data from AHB_monitor via analysis port when write() function is called in AHB_monitor. Different covergroups are defined as per verification plan. Covergroups are instantiated in new() function and sampled on receiving each packet from AHB_monitor.

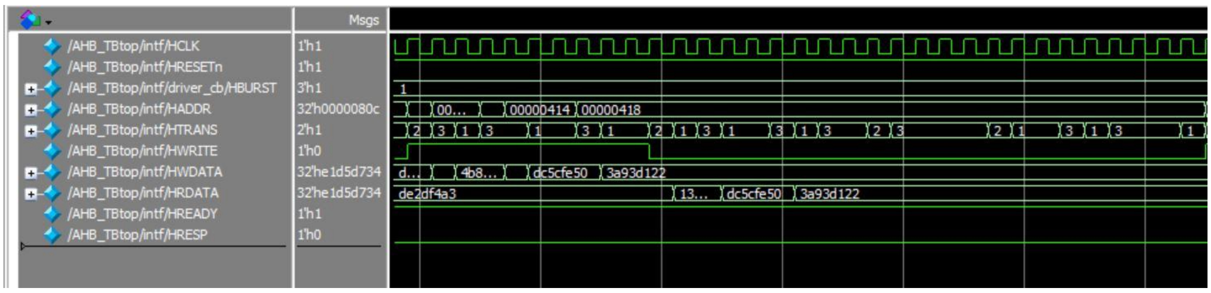
4. Results and Analysis

4.1 Waveforms



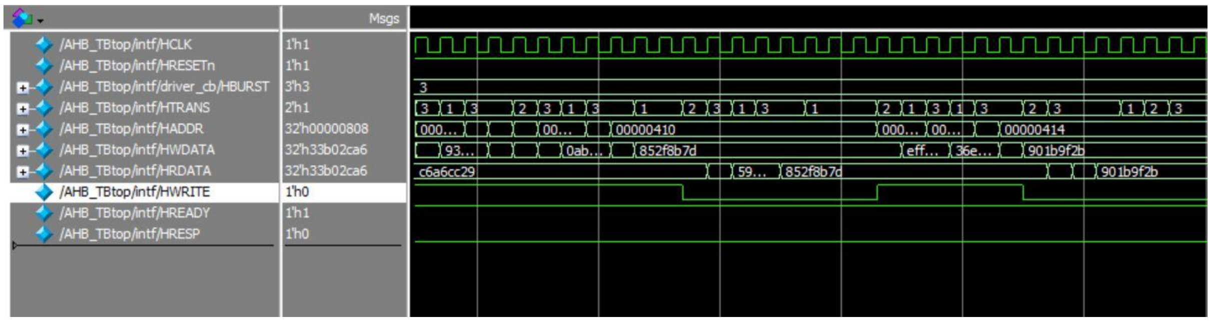
```
/AHB_TBtop/intf/HCLK                 1'h1
/AHB_TBtop/intf/HRESETn              1'h1
/AHB_TBtop/intf/driver_cb/HBURST     3'h0
/AHB_TBtop/intf/HTRANS               2'h2
/AHB_TBtop/intf/HADDR                32'h000007fc
/AHB_TBtop/intf/HWDATA               32'h86f85552
/AHB_TBtop/intf/HRDATA               32'hxxxxxxxx
/AHB_TBtop/intf/HWRITE               1'h0
/AHB_TBtop/intf/HREADY               1'h1
/AHB_TBtop/intf/HRESP                1'h0
```

SINGLE burst transfer



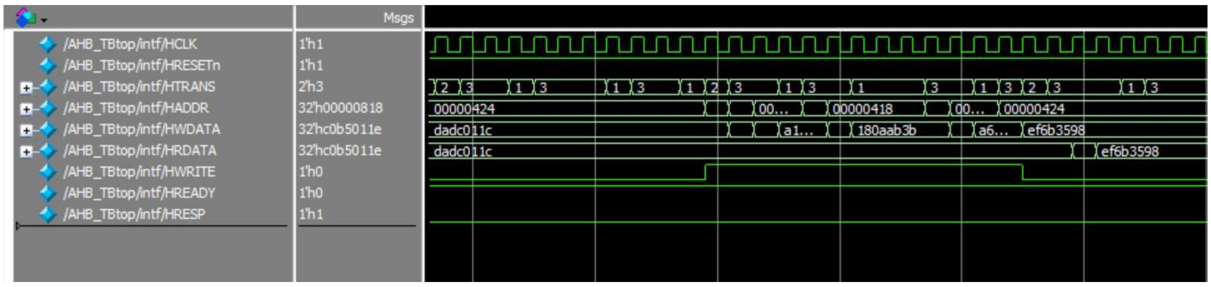
```
/AHB_TBtop/intf/HCLK                 1'h1
/AHB_TBtop/intf/HRESETn              1'h1
/AHB_TBtop/intf/driver_cb/HBURST     3'h1
/AHB_TBtop/intf/HADDR                32'h0000080c
/AHB_TBtop/intf/HTRANS               2'h1
/AHB_TBtop/intf/HWRITE               1'h0
/AHB_TBtop/intf/HWDATA               32'he1d5d734
/AHB_TBtop/intf/HRDATA               32'he1d5d734
/AHB_TBtop/intf/HREADY               1'h1
/AHB_TBtop/intf/HRESP                1'h0
```

INCR burst transfer



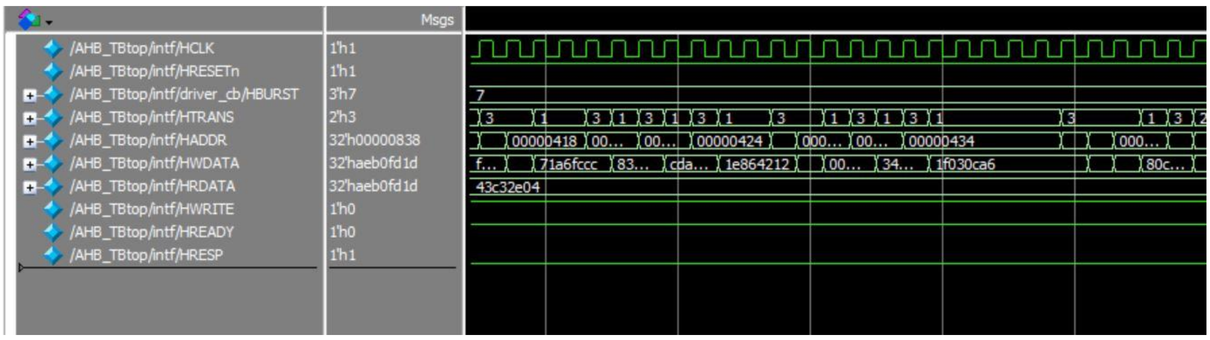
```
/AHB_TBtop/intf/HCLK                 1'h1
/AHB_TBtop/intf/HRESETn              1'h1
/AHB_TBtop/intf/driver_cb/HBURST     3'h3
/AHB_TBtop/intf/HTRANS               2'h1
/AHB_TBtop/intf/HADDR                32'h00000808
/AHB_TBtop/intf/HWDATA               32'h33b02ca6
/AHB_TBtop/intf/HRDATA               32'h33b02ca6
/AHB_TBtop/intf/HWRITE               1'h0
/AHB_TBtop/intf/HREADY               1'h1
/AHB_TBtop/intf/HRESP                1'h0
```

INCR4 burst transfer



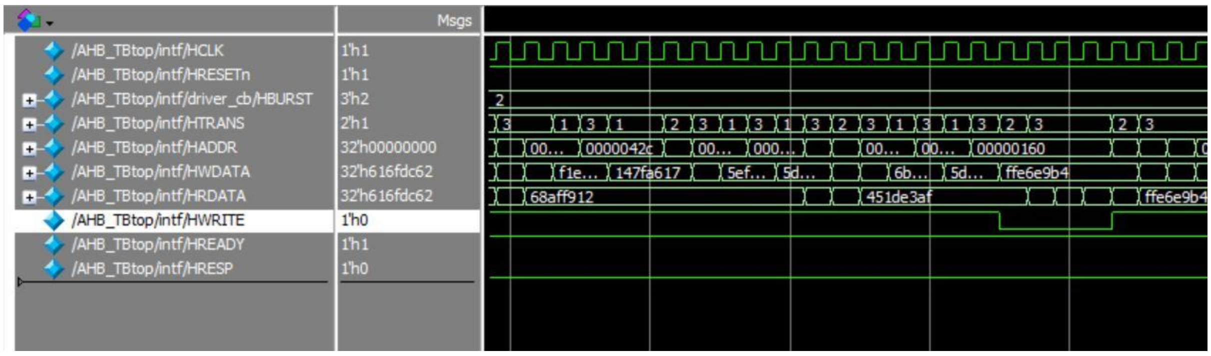
```
/AHB_TBtop/intf/HCLK                 1'h1
/AHB_TBtop/intf/HRESETn              1'h1
/AHB_TBtop/intf/HTRANS               2'h3
/AHB_TBtop/intf/HADDR                32'h00000818
/AHB_TBtop/intf/HWDATA               32'hc0b5011e
/AHB_TBtop/intf/HRDATA               32'hc0b5011e
/AHB_TBtop/intf/HWRITE               1'h0
/AHB_TBtop/intf/HREADY               1'h0
/AHB_TBtop/intf/HRESP                1'h1
```

INCR8 burst transfer



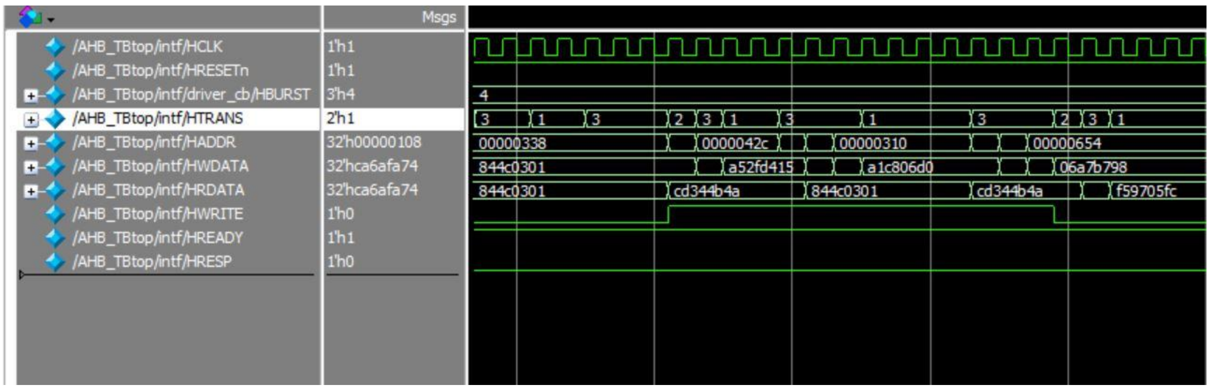
```
/AHB_TBtop/intf/HCLK                1'h1
/AHB_TBtop/intf/HRESETn             1'h1
/AHB_TBtop/intf/driver_cb/HBURST    3'h7
/AHB_TBtop/intf/HTRANS              2'h3
/AHB_TBtop/intf/HADDR               32'h00000838
/AHB_TBtop/intf/HWDATA              32'haeb0fd1d
/AHB_TBtop/intf/HRDATA              32'haeb0fd1d
/AHB_TBtop/intf/HWRITE              1'h0
/AHB_TBtop/intf/HREADY              1'h0
/AHB_TBtop/intf/HRESP               1'h1
```

INCR16 burst transfer



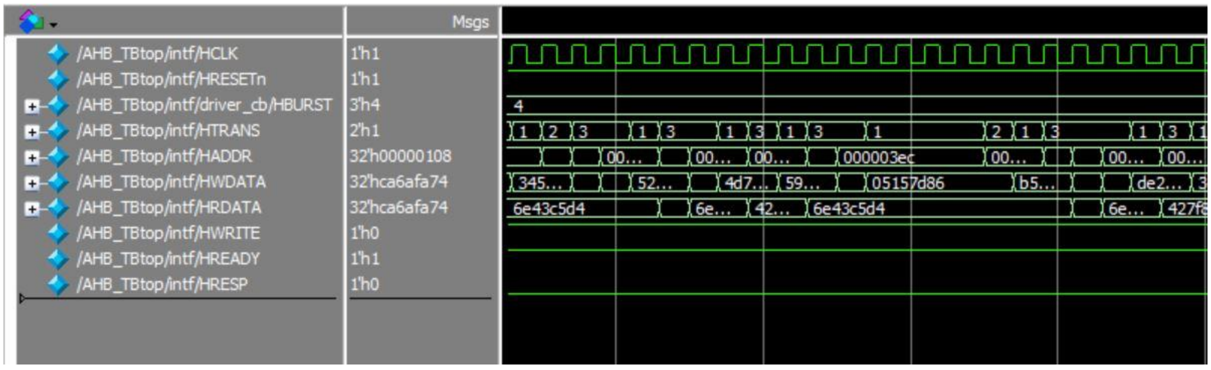
```
/AHB_TBtop/intf/HCLK                1'h1
/AHB_TBtop/intf/HRESETn             1'h1
/AHB_TBtop/intf/driver_cb/HBURST    3'h2
/AHB_TBtop/intf/HTRANS              2'h1
/AHB_TBtop/intf/HADDR               32'h00000000
/AHB_TBtop/intf/HWDATA              32'h616fdc62
/AHB_TBtop/intf/HRDATA              32'h616fdc62
/AHB_TBtop/intf/HWRITE              1'h0
/AHB_TBtop/intf/HREADY              1'h1
/AHB_TBtop/intf/HRESP               1'h0
```

WRAP4 burst transfer



```
/AHB_TBtop/intf/HCLK                1'h1
/AHB_TBtop/intf/HRESETn             1'h1
/AHB_TBtop/intf/driver_cb/HBURST    3'h4
/AHB_TBtop/intf/HTRANS              2'h1
/AHB_TBtop/intf/HADDR               32'h00000108
/AHB_TBtop/intf/HWDATA              32'hca6afa74
/AHB_TBtop/intf/HRDATA              32'hca6afa74
/AHB_TBtop/intf/HWRITE              1'h0
/AHB_TBtop/intf/HREADY              1'h1
/AHB_TBtop/intf/HRESP               1'h0
```

WRAP8 burst transfer



```
/AHB_TBtop/intf/HCLK                1'h1
/AHB_TBtop/intf/HRESETn             1'h1
/AHB_TBtop/intf/driver_cb/HBURST    3'h4
/AHB_TBtop/intf/HTRANS              2'h1
/AHB_TBtop/intf/HADDR               32'h00000108
/AHB_TBtop/intf/HWDATA              32'hca6afa74
/AHB_TBtop/intf/HRDATA              32'hca6afa74
/AHB_TBtop/intf/HWRITE              1'h0
/AHB_TBtop/intf/HREADY              1'h1
/AHB_TBtop/intf/HRESP               1'h0
```

WRAP16 burst transfer

4.2 Test Results

• Total Transfer Count	= 4404
• Successful Transfer Count	= 4079
• Failed Transfer Count	= 325
• Successful Transfer Percentage	= 92.620345 %
• Failed Transfer Percentage	= 7.37965 %

4.3 Failed Test cases

- INCR16 Write
- INCR16 Read
- WRAP16 Write
- WRAP16 Read
- All read – write sequences defined for INCR16 and WRAP16

4.4 Coverage Results

4.4.1 Functional coverage:

4.4.1.1 Statistics:

Total Functional Coverage = 88.697 %
Coverage For Ahb_Functional_Coverage Covergroup = 98.076 %
Coverage For Sequence_Of_Operations_Coverage Covergroup = 79.381 %

4.4.1.2 Analysis:

Ahb_Functional_Coverage Covergroup:

- Read – Read to different addresses uncovered

Sequence_Of_Operations_Coverage Covergroup:

- Read – Read to different addresses for all burts are uncovered
- Write – Read – Write to same address for all bursts uncovered
- Read – Write – Read to same address for all bursts uncovered
- Write – Write – Read to same address for all bursts uncovered

Note: Due to the time constraint, the above cases were uncovered, but can be covered by adding additional tests and updating the cross bins

4.4.2 Code coverage:

4.4.2.1 Statistics:

Total Code Coverage = 88.28 %

AHBSlave:

Enabled Coverage	Coverage %
Branches	90.90
Conditions	75.00
Statements	87.50
Toggles	83.33

AHBSlaveTop:

Enabled Coverage	Coverage %
Branches	100.00
Conditions	100.00
Statements	100.00
Toggles	79.31

4.4.2.2 Analysis:

- AHBSlave operates in two different modes, error inject mode and without error mode**,** it can be operated only in one mode at a time, hence code coverage for (Branches, Conditions, Statements) in a single run is not 100 %
- HADDR [31:11] are zero in this present test suite , hence the toggle coverage for HADDR is not 100 %

4.5 Transcript

```
# <<FUNCTIONAL COVERAGE>>
# TOTAL FUNCTIONAL COVERAGE = 88.697552
# COVERAGE FOR AHB_functional_coverage COVERGROUP = 98.076923
# COVERAGE FOR sequence_of_operations_coverage COVERGROUP = 79.318182
#
#
# UVM_INFO AHB_scoreboard.sv(150) @ 44035: uvm_test_top.env.scoreboard
[AHB_scoreboard]
# TOTAL TRANSFER COUNT = 4404
# TRANSFER SUCCESS COUNT = 4079
# TRANSFER FAILURE COUNT = 325
# TRANSFER PASS PERCENTAGE = 92.620345
# TRANSFER FAIL PERCENTAGE = 7.379655
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :  331
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [AHB_coverage]     1
# [AHB_scoreboard]   326
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# ** Note: $finish    : C:/questasim64_2020.3/win64/../verilog_src/uvm-
1.1d/src/base/uvm_root.svh(430)
#    Time: 44035 ns  Iteration: 67  Instance: /AHB_TBtop
```

Note: Above transcript shows only the reports, detailed file is provided as in attachments.

5. References

5.1 References:

- AMBA AHB-LITE Specification
- Dr. Tom Schubert Lecture Slides
- Comprehensive Functional Verification – Bruce Wile, John C. Goss, W Rosner UVM Primer – Ray Salemi
- https://www.chipverify.com/uvm/uvm-tutorial
- https://verificationguide.com/uvm/uvm-tutorial/
- https://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf
- UVM Rapid Adoption: A Practical Subset of UVM at https://s3.amazonaws.com/verificationacademy-news/DVCon2015/Papers/dvcon-2015_UVM-Rapid-Adoption-A-Practical-Subset-of-UVM-Paper.pdf
- Incisive coverage Introduction and RAK overview at http://docshare01.docshare.tips/files/31703/317032353.pdf
- OVM/UVM Scoreboards - Fundamental Architectures at http://sunburst-design.com/papers/CummingsSNUG2013SV_UVM_Scoreboards.pdf

6. Attachments

6.1 RTL

- AHBSlaveTop.sv
- AHBSlave.sv
- definespkg.sv

The above files can be found in the below link - RTL-Source

6.2 TestBench

- AHB_interface.sv
- AHB_pkg.sv
- AHB_sequence_item.sv
- AHB_sequences.sv
- AHB_sequencer.sv
- AHB_virtual_sequence.sv
- AHB_virtual_sequencer.sv
- AHB_driver.sv
- AHB_monitor.sv
- AHB_agent.sv
- AHB_packet.sv
- AHB_coverage.sv
- AHB_scoreboard.sv
- AHB_environment.sv
- AHB_test.sv
- AHB_TBtop.sv

The above files can be found in the below link –

Github-Group3-AMBA-AHB-Lite/UVM/TestBench/Portland_State_University