AMBA AHB-Lite Slave Protocol Verification Plan

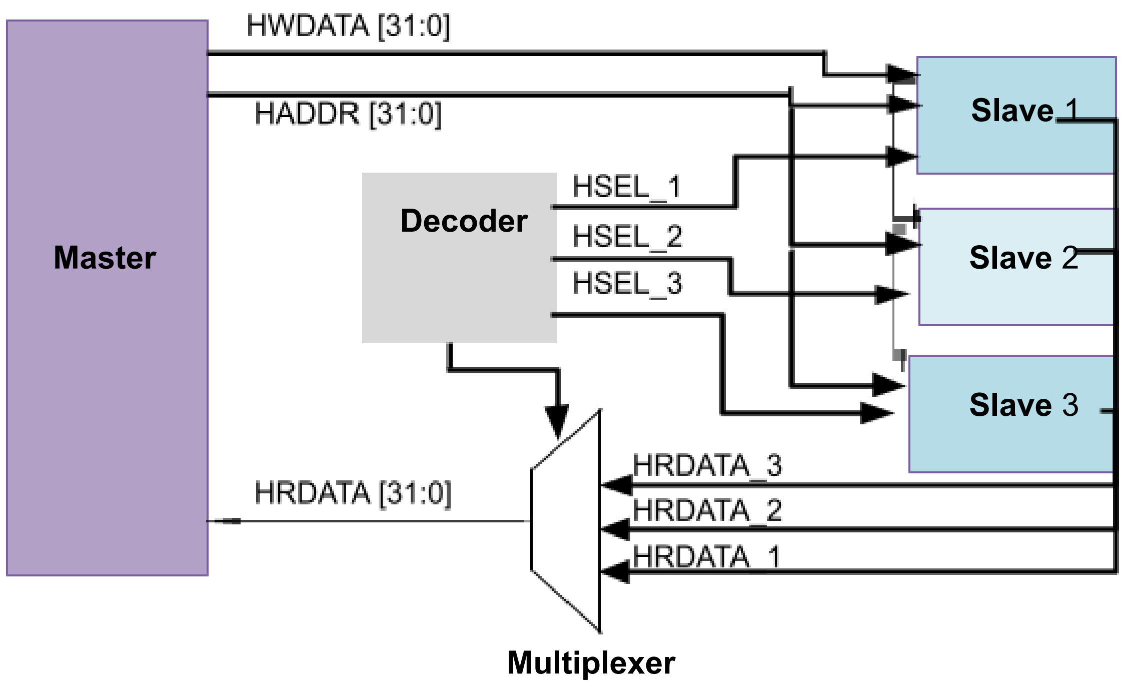
May 18, 2023.

Submitted By:

Wajahat Riaz

AHB-Lite Protocol:

Advance High Performance-Lite (AHB-lite) is a bus interface that supports a single bus master and provides high bandwidth operations. The most common slaves used for this protocol are internal memory devices, external memory interfaces, and high-bandwidth peripherals. The figure below illustrates an AHB-Lite system design with one master and multiple slaves.



The main components of the AHB-Lite system are as follows:
1) Master
2) Slave
3) Decoder
4) Multiplexor

An AHB-Lite master provides address and control information to initiate read and write operations. The slave responds to transfers initiated by masters in the system. The slave uses the select signal from the decoder to control when it responds to a bus transfer. The slave signals back to the master i.e., the success, failure, or waiting of the data transfer. 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 slave-to-master multiplexor is required to multiplex the read data bus and response signals from the slaves to the master.

Working of Protocol:

The master starts a transfer by driving the address and control signals. These signals provide information about the address, direction, and width of the transfer, and indicate if the transfer forms part of a burst. Transfers can be of different types for instance single, incrementing bursts that do not wrap at address boundaries, wrapping bursts that wrap at

particular address boundaries, etc. The write data bus moves data from the master to a slave, and the read data bus moves data from a slave to the master.
Every transfer consists of two phases:
1) Address phase: one address and control cycle
2) Data phase: one or more cycles for the data.

A slave cannot request that the address phase is extended and therefore all slaves must be capable of sampling the address during this time. However, a slave can request that the master extends the data phase by using an HREADY signal. This signal, when LOW, causes wait states to be inserted into the transfer and enables the slave to have extra time to provide or sample data. The slave uses a response signal to indicate the success or failure of a transfer.

Global Signals:

Name	Destination	Description
HCLK	Clock Source	The clock source for all operations on the protocol. Input signals are sampled at the rising edge and changes in output signals happen after the rising edge
HRESTn	Reset Controller	Asynchronous primary reset for all bus elements

Master Signals:

Name	Destination	Description
HADDR [31:0]	Slave and Decoder	Address bus of 32 bits
HBURST [2:0]	Slave	Indicates the type of burst signal including wrapping and incrementing bursts
HSIZE [2:0]	Slave	Indicates the size of transfer from 8 bits to 1024 bits
HPROT [3:0]	Slave	Protection control signal providing additional information like opcode fetch, data fetch, etc.

Slave Signals:

Name	Destination	Description

HRDATA [31:0]	Multiplexor	Read data bus to transfer the data from a Slave's location to the Master via multiplexor
HREADYOUT	Multiplexor	Indicates transfer has finished on the bus and is used to extend the data phase
HRESP	Multiplexor	Provides additional information on whether the transfer was successful or failed

Decoder Signals:

Name	Destination	Description
HSELx Note: x is a unique identifier for AHB lite slave	Slave	Indicates current transfer is intended for the selected slave

Multiplexor Signals:

Name	Destination	Description
HRDATA [31:0]	Master	Read data bus to rout to Master
HREADY	Master and Slave	Indicates completion of previous transfer
HRESP	Master	Transfer response

Software/Tools:

1) VCS Synopsys
2) Gtkwave
3) Google Docs

Part A - Create a Verification Plan

- Read and understand the protocol specifications
- Write all the features which need to tested for verification of the DUT
- Write all the different test cases which need to be created for each specific feature
- Write the required sequences/stimulus for performing each test case
- After the verification is done, report the results for each test case, use the template provided with the task.
- Report the coverage. Coverage is an important metric that indicates which parts of
- design/ code got covered and which were not. [Hint: You can use the VCS user guide to learn how to measure the coverage of your project.
- Get as close to 100% coverage as possible by writing/adding the additional sequences/stimulus that you need
- Get the bugs fixed and repeat.

Verification Plan:

No.	Feature	Test Description	Ref.	Type	Result	Expected Outcome	Comments
1	Write transfer to slave	An address A is driven onto the bus. The slave will sample the address A on the next rising clock edge. Afterward, the slave will drive the HREADY response. This response is sampled on the next rising edge of HCLK.	3.1	TR	PASS	The address phase should not be more than one cycle. The slave must only sample address when HREADY is high. The Data(A) must be written at the address A and a completed transfer is signaled i.e., HRESP should be low and HREADY should be high.	N/A

						The address and data phases must be separate	
2	Read transfer from slave	An address A is driven onto the bus. The slave will sample the address A on the next rising clock edge. Afterward, the slave will drive the HREADY response. This response is sampled on the next rising edge of HCLK.	3.1	TR	FAIL	The address phase should not be more than one cycle. The slave must only sample address when HREADY is high. The Data(A) must be fetched from the address A and a completed transfer should be signaled i.e. HRESP should be low and HREADY should be high. The address and data phases must be separate.	Read the transfer does not have a separate data phase.
4	Read followed Write transfer at a fixed address	Fixed address A is driven onto the bus for this test case. The slave will sample the address on the rising edge of the clock provided that HREADY is high. After sampling the address. Wait for states to be added in the data phase by keeping HREADY low for two cycles after we have sampled the address	3.1 5.1 5.1.2	A	FAIL	write transfer should follow test case 1 and read transfer should follow test case 2.	Read transfer does not have a separate data phase.
5	Protection signals HPROT [3:0]:	The protection signal HPROT [3:0] = 4'b0001 corresponds to	3.7	A	PASS	The protection signal gives extra information which	N/A

	• 4’b0001	non-cacheable, non-bufferable, unprivileged data access only.				can be used to determine an exception for instance illegal instruction, illegal access, etc. For instance, data(A) can’t be accessed because only a privileged level can access that information. The response is entirely dependent on how the design engineer implemented it.	
6	Global Signal: HRESTn	Before continuous read transfers an HRESTn is driven low asynchronously.	7.1.2	A	PASS	All bus elements will initialize to valid levels. Read transfers will read initialized values from the slave memory	N/A
7	Master Signal: IDLE HTRANS [1:0] =b00	An address A is driven onto the bus. An IDLE transfer is inserted to this address.	3.2	A	PASS	The transfer must be ignored by the slave. Slave provides a zero-wait OKAY response.	N/A
8	Master Signal: BUSY HTRANS [1:0] =b01	An address A is driven onto the bus. A BUSY transfer is inserted to this address.	3.2	A	PASS	The transfer must be ignored by the slave since we have only single burst transactions Slave provides a zero-wait OKAY response.	N/A

9	Transfer type changes from IDLE to NONSEQ	Address A and B are driven onto the bus. Half transactions are IDLE and the other half is NONSEQ implemented in interleaved manner	3.6.1	A	PASS	The slave will sample address A at the rising clock edge of the address phase. After a successful transfer to address A the slave will ignore the IDLE transfers i.e., transfers associated with address B will be neglected.	
10	Slave response: Transfer done	A write or read transfer is carried out on address A.	5.1.1	A	PASS	A completed transfer is signaled when HREADY is high and HRESP is OKAY.	This was an implicit test in the previous tests
11	Slave response: ● Transfer done ● Transfer pending	An address A is driven onto the bus with 3 waited-for states. This is done by keeping the HREADY low during the data phase of address A. Then an address B is driven onto the bus with zero wait states. The HRESP and HREADY signals are monitored	5.1.1 5.1.1	A	PASS	During the waited states, the HRESP is low and HREADY is also low which indicates that the transfer is pending. After the wait states, the transfer of address A is signaled i.e., HRESP is high and HREADY is high.	These assertions were implicit in the previous tests.
12	Slave response: HSEL = 0	Performing a read or write transfer given that slave is not selected during transactions	5.1.3	A	PASS	The effect of transfers has no effect on the DUT since it’s not connected. No response is given	

Explanation of Different Fields

No.	The serial number of the test.
Feature	The feature which the current test is verifying in full or partially. The feature is usually on the abstraction level of a user.
Test Description	A detailed description of the test case is performed. You can be as verbose as you want.
Ref.	Reference to the section in the related standard document. The section number, as well as page numbers, should be described here.
Type	Type of the test. Whether the test is an assertion (A) or a transaction (T) type.
Result	Indicates that the given module has passed or failed the test.
Comments	Any other comments about the test or its results that you want to mention?

Part B – Modifying your verification environment

The hierarchy of your TestBench environment is expected to remain unchanged, although you are free to experiment, try your ideas, and innovate. Following are some of the questions that you might ask yourself along with many others:

Transaction:

- Can I run predefined sequences? (e.g. reset sequence, random write sequence, random read/write bursts, a directed test).
  YES.

- Can I debug easily if my test fails?

YES. Using the print_trans() method of the transaction class.

- Do I need one or multiple transactions for bursts?
  Not sure what multiple transactions mean. The test cases were written for **n** number of transactions. However, for a single burst, we only require one transaction.

Generator:

- How to control how many transactions get generated? Sometimes random transactions are not needed.
  Firstly, we are keeping track of our transactions locally in our classes. The number of local transactions is incremented after every iteration until the max number of transactions. The max number of transactions is provided by the program block. Within the environment we are waiting for the events concurrently given below:

```
fork
        wait(drv.local_count == no_transaction);
        wait(mon.local_count == no_transaction);
        wait(scor.local_count== no_transaction);
join
```

- How do I generate non-random transactions when required?
  In order to generate non-random transactions, first we need to disable the randomization in the tes.sv file using the rand_mode(arg) method. This depends entirely on the test case.

Driver

- Are the interface signals driven according to the spec?
  Yes, all the interface signals are driven according to the specifications.

- Does the transaction have the proper address/data Phases?

The transactions involving a write transfer follow the specification. The address and data phases are separate. Whereas for transactions involving read transfers, the design does not follow the specification. This was noticed when implementing a test involving read-write transactions in an interleaved manner.

- Do I need to sample inputs to decide whether to drive outputs or not on the next clocking event?
  No, you are not required to do so. However, this is a common practice since we want our input signals to be stable and valid.

Monitor

- Are the interface signals sampled according to the spec?
  Yes, they are sampled according to the specifications.

- Does the transaction have proper address/data Phases?
  The transactions involving a write transfer follow the specification. The address and data phases are separate. Whereas for transactions involving read transfers, the design does not follow the specification. This was noticed when implementing a test involving read-write transactions in an interleaved manner.

Scoreboard:

- Does the scoreboard implement proper endianness?
  Yes, the scoreboard that I implemented takes care of the endianness as well. I have only implemented endianness for write transactions. Implementation for read transfers was buggy.

- How to change endianness if required?
  Well, you can't change the design.sv file. However, if there was a bit provided by the slave DUT we can signal out the scoreboard to implement the corresponding endianness.

- How to not compare reset values and compare only those memory locations which have already been written?
  This is actually hard. We need to provide some metadata. We require a single bit to signal whether a particular memory location was written before or not.

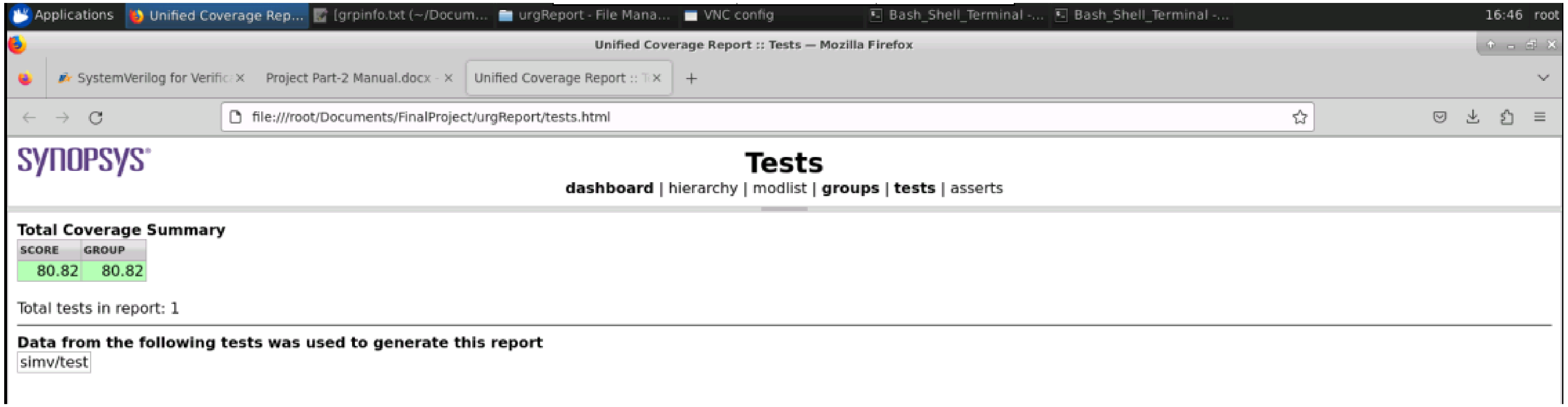
- Should scoreboard memory be static or dynamic?
  The scoreboard memory should be static. This is because of a number of reasons.

Coverage Report:

Type the following command to generate a coverage report:

```
urg -lca -dir simv.vdb
```

For a single test I have achieved 80% coverage.


Applications
Unified Coverage Rep...
[grpinfo.txt (~/Docum...
urgReport - File Mana...
VNC config
Bash_Shell_Terminal -...
Bash_Shell_Terminal -...
16:46 root
Unified Coverage Report :: Tests — Mozilla Firefox
SystemVerilog for Verific
Project Part-2 Manual.docx
Unified Coverage Report :: T
file:///root/Documents/FinalProject/urgReport/tests.html
SYNOPSYS
dashboard | hierarchy | modlist | groups | tests | asserts

Tests

Total Coverage Summary

SCORE	GROUP
80.82	80.82

Total tests in report: 1

Data from the following tests was used to generate this report

simv/test