ECE 571
Introduction to SystemVerilog
Winter 2018

Design and Verification Environment to verify ARM based AMBA AHB-Lite protocol Master and Slave

Submitted by:

Kirtan Mehta
Mohammad Suheb Zameer
Raveena Khandelwal
Sai Tawale

INDEX

PROPOSED PROJECT:

● **DUT:** We are going to design a synthesizable DUT which includes the design of one master and one slave. We checked the functionality of the code as per the specification of AHB lite.

● **Test Plan:** Check the basic functionality of the AHB-Lite DUT. Randomize the inputs with some constraints and check the corresponding outputs based on test cases for different scenarios while keeping variation in inputs, reset and parameters. Injection scenarios in the design and the test plan should be able to point out the errors accordingly.

● **Test Environment:** The test environment would be based on the OOPS concept i.e classes. As a part of the test environment, we will be writing separate classes to perform specific operation like stimulus generator , drivers, monitoring, checker, scoreboard, test and top.

● **Debugging:** If the environment compiles and there are no errors, as a part of debugging, we will Provide coverage, check all the signals if they are working as per the protocol. We will check if there are Any logical errors and for all the test cases, we will debug accordingly. Using display statements,

Waveform etc would be one of the many ways to debug the code.

● **Emulation**: Emulation would be done in TBx mode where the design will run on veloce and testbench Will run on Questa Simulator.

Extension to the project: If time permits, we will also do multi slave in DUT and work on it's Debugging. This will give us the verification of the entire AMBA AHB-Lite protocol. For now, we are Designing and verifying single master and single slave in AHB-Lite using a class based verification Environment.

We will design the single master and single slave and then verify but still we are going to build class based SystemVerilog environment to verify it.

TARGETS ACHIEVED

We have designed master and slave with basic functionality for write operation by master and read operation by slave based on HSIZE, HTRANS and HBURST. We have also made a basic testbench to see if the DUT is working or not.

After this, we designed the whole environment based on randomization and constraints. On this, we have verified the code for its address, i.e whatever the address, we check the data for the same.

Lastly, we have performed emulation in TBX mode

BRIEF OVERVIEW OF CLASSES IN SYSTEMVERILOG:

SystemVerilog introduces an object oriented class datatype. Classes allow objects to be dynamically created, deleted, assigned, and accessed via object handles. A class is a type that includes data and subroutines that operate on that data. An object is an instance of a particular class. An object is defined by using a handler of class type and then calling new function which allocates memory to the object.

FUNDAMENTAL PRINCIPLES OF OOPS:

1. ENCAPSULATION: It is a concept that binds together data and functions that manipulate the data. Encapsulation keeps both the data and functions safe from outside world, i.e data hiding.

2. ABSTRACTION: Abstraction is the concept of moving the focus from the details and concrete implementation of things, to the types of things, the operations available, thus making programming more simple and general.
3. INHERITANCE: New classes are created by inheriting properties and method defined in existing class. Existing class is called the base class (parent class), and the new class is referred to as the derived class (child class).
4. POLYMORPHISM: Polymorphism means having many forms. A member function will cause a different function to be executed depending on the type of object that invokes the function.

BRIEF OVERVIEW OF ASSERTIONS IN SYSTEMVERILOG:

Assertions are checks which are used to verify that your design meets the given requirements. Coverage are used to judge what percentage of your test plan or functionality has been verified. They are used to judge the quality of the stimulus. They help us in finding what part of the code remains untested. There are two types of assertions: Immediate assertions and Concurrent assertions.

Immediate assertions are used to check condition at current time. These checks are non-temporal, i.e not performed across time or clock cycles. They are used inside procedural blocks (initial/ always and tasks/functions).

Concurrent assertions test for a series of events spread over multiple clock cycles, i.e they are temporal in nature. “property” keyword is used to define concurrent assertions. Property is used to define a design specification that needs to be verified. They are called concurrent because they occur in parallel with design blocks.

BRIEF OVERVIEW OF AMBA 3 AHB-Lite 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.

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

- burst transfers
- single-clock edge operation

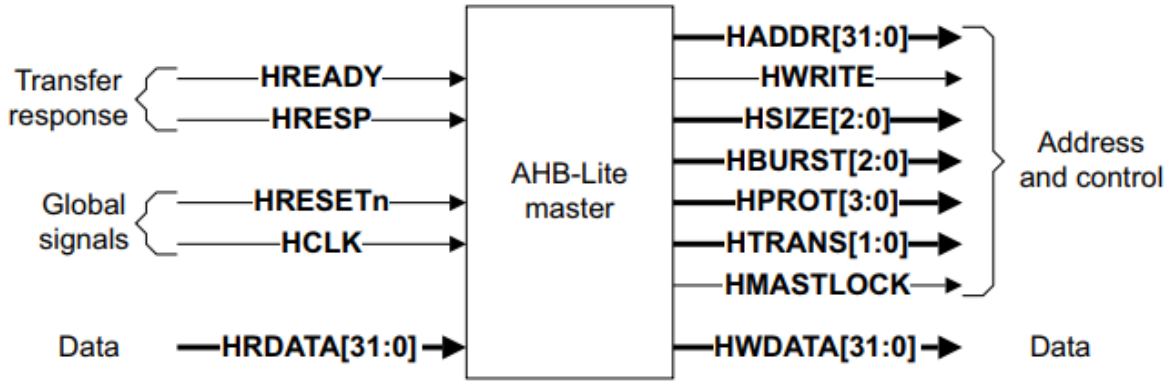
- non-tristate implementation
- wide data bus configurations, 64, 128, 256, 512, and 1024 bits.

The most common AHB-Lite slaves are internal memory devices, external memory interfaces, and high bandwidth peripherals. Although low-bandwidth peripherals can be included as AHB-Lite slaves, for system performance reasons they typically reside on the AMBA Advanced Peripheral Bus (APB). Bridging between this higher level of bus and APB is done using AHB-Lite slave, known as an APB bridge.

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.

MASTER:

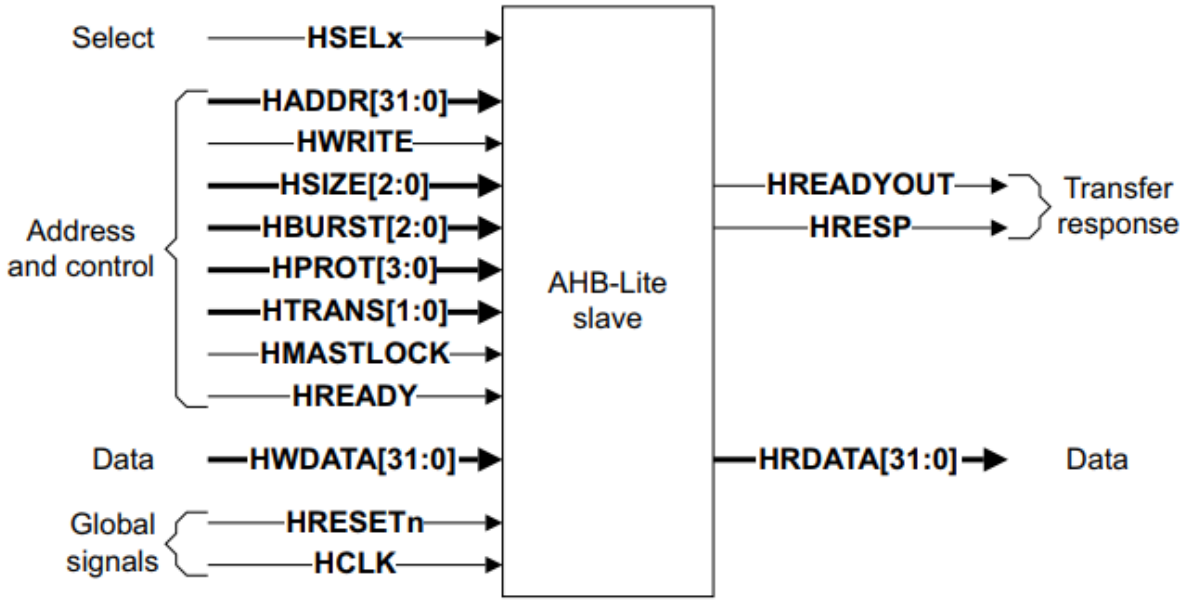
An AHB-Lite master provides address and control information to initiate read and write operations.



SLAVE:

An AHB-Lite 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. The slave signals back to the master:

- The success
- failure
- waiting of the data transfer.



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.

MULTIPLEXER:

A slave-to-master multiplexor is required to multiplex the read data bus and response signals from the slaves to the master. The decoder provides control for the multiplexor.

BASIC OPERATION OF AMBA 3 AHB-LITE PROTOCOL

The master starts a transfer by driving the address and control signals. These signals provide information about the address, direction, width of the transfer, and indicate if the transfer forms part of a burst. Transfers can be:

• Single

• Incrementing bursts that do not wrap at address boundaries

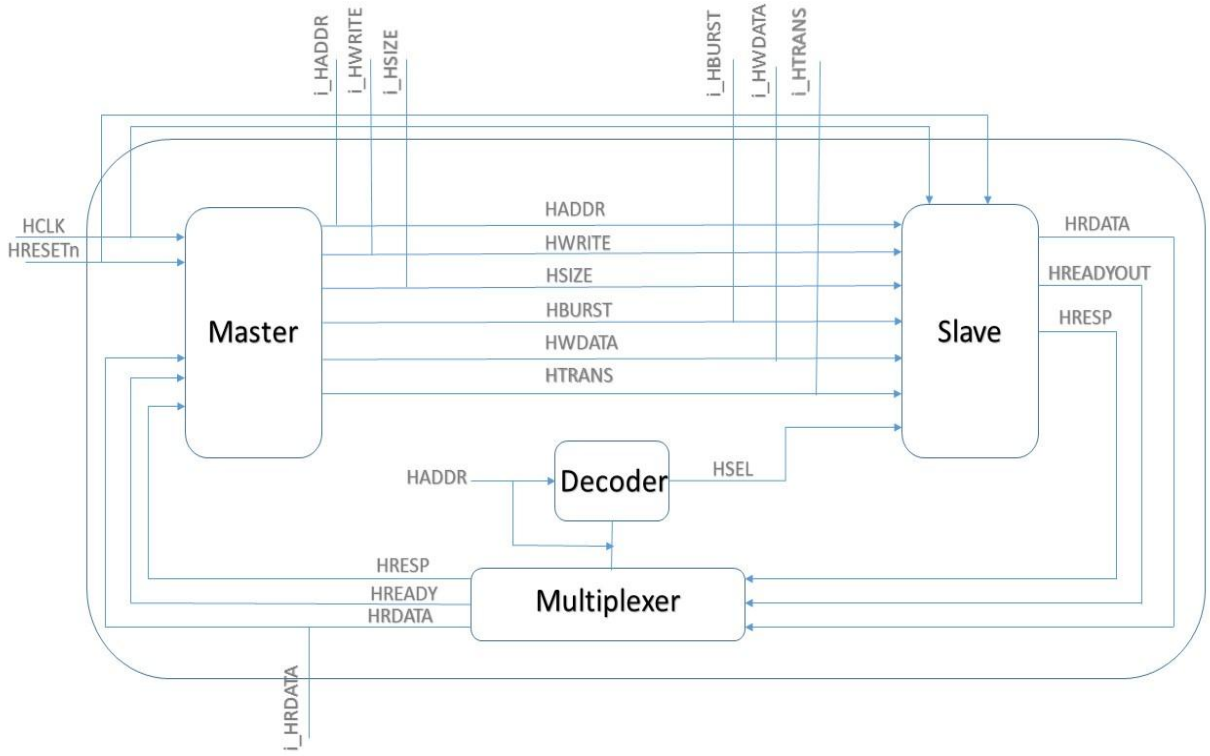
• Wrapping bursts that wrap at particular address boundaries.

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: Address phase one address and control cycle 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 HREADY. 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 HRESP to indicate the success or failure of a transfer.

VERIFICATION STRATEGY:

1. Development of test cases and interfaces and integrate them with DUT in the top module.
2. Development of the environment class.
3. Development of packet class based on stimulus plan.
4. Development of the driver class. Packets are generated and sent to DUT using driver.
5. Development of receiver class. Receiver collects packets coming from the output port of DUT.
6. Development of coverage class based on coverage plan. In this phase, we will test and analyze coverage report.

DESIGN SPECIFICATION:

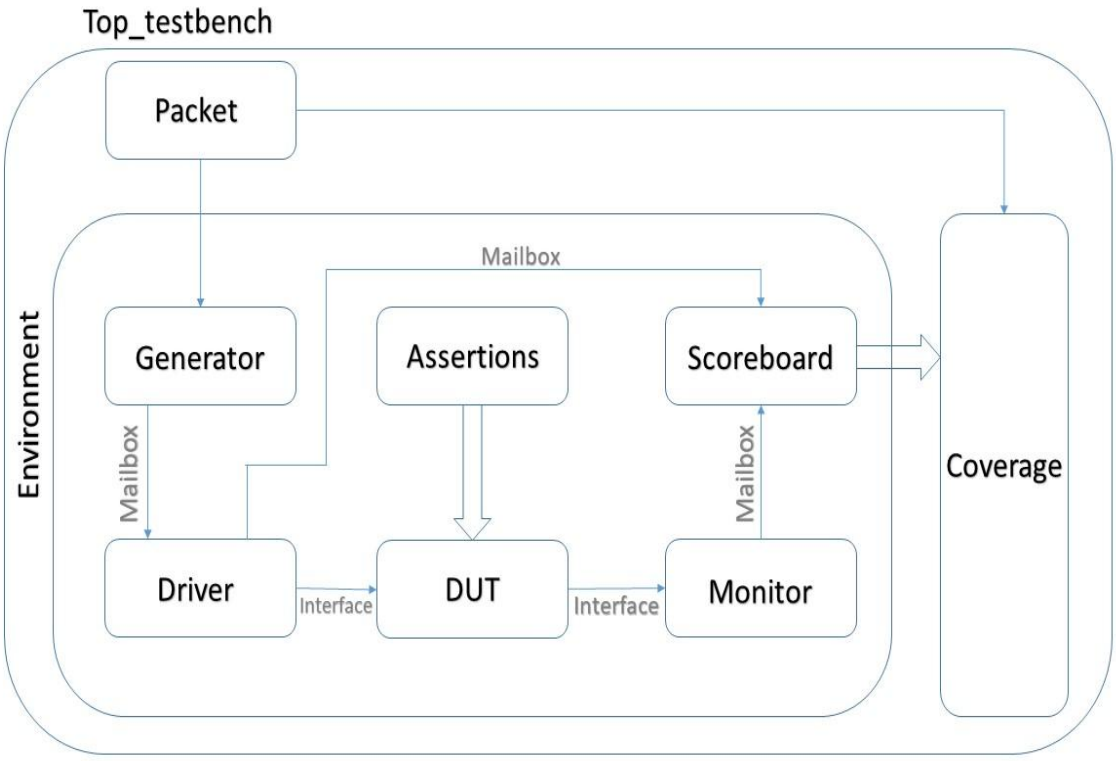


In our design we have one Master and one Slave. Signals travelling from Master to Slave are the HADDR, HWRITE, HSIZE, HBURST, HWDATA and HTRANS. Both the master and the slave get the clock and reset signals. The decoder provides select lines for the selection of slaves. In our design, we have just one select line from the decoder to the slave. When this line is activated, the master communicates with the slave. Output of the Slave is HRDATA, HREADYOUT and HRESP. These signals are going to the Multiplexer and in return, the Multiplexer provides signals to the Master. These are the HRESP, HREADY and HRDATA. This block is our top design module. From outside this rectangular block, we are providing external input and output signals such as i_HADDR, i_HWRITE, i_HSIZE, i_HBURST, i_HWDATA and i_HTRANS and the output signal is the i_HRDATA. The clock and reset are also the external signals to this top design block.

HADDR is the address which is 32 bits wide. HWDATA is 32 bit data to be written. As soon as the HWDATA signal goes high, the data gets written on the address that we are giving. HSIZE, HTRANS and HBURST are different modes. Idle, synchronous, nonsynchronous data etc, depends on the value of HTRANS which varies from 00 to 11. HBURST determines the size of the burst that we are sending. It tells us whether it is an incremental burst or wrap around burst. HSIZE is the size of the burst like 4, 8, 16 etc. For our code, we are sending 32 bits of burst size.

Once a particular value is written on the slave, if HRESP is high, then we have an error. If it is low, i.e 0, it is an error and we have to send the packet again. If an error is present, it will take 2 cycles to execute.

If HREADY is 1, that means that the transfer is pending, otherwise it is successful. If the transfer is pending, then additional cycles are required to complete the cycle successfully.



ENVIRONMENT :

Packet, Generator and driver are our different classes. DUT is the design under test. From the classes and DUT, we go to the monitor and scoreboard. Driver to DUT and DUT to Monitor is our Interface. Remaining arrows are the mailbox. The solid arrows signify the data that transferred between the assertions and DUT and Scoreboard to the coverage. We take inputs from the packet and scoreboard and check the coverage for the same.

PHASE 1: TOP

We wrote the SystemVerilog interfaces for input port, output port and memory port. Top module test cases and DUT instances are done. DUT and Testbench interfaces are connected to top module. Clock and reset signals are also generated in the top module.

In the interface file, all the interfaces have clock as the input. All the signals within the interface are of logic type. All the signals are synchronized to clock except reset in clocking block. Signal direction WRT TestBench is specified with modport.

PHASE 2: ENVIRONMENT

In this phase, we write the environment class, virtual interface declaration, defining environment class constructor, and defining required methods for execution. This class is the base class used to implement verification environments. Testcase contains the instance of the environment class and has access to all the public declaration of environment class. All methods are declared as virtual methods. In environment class, we will formalize the simulation steps using virtual methods. The methods are used to control the execution of the simulation.

Connecting the virtual interfaces of Environment class to the physical interfaces of top module. Verification environment contains the declarations of the virtual interfaces. Virtual interfaces are just a handles (like pointers). When a virtual interface is declared, it only creates a handle. It does not create a real interface.

PHASE 3: RESET

In this phase, we reset and configure the DUT.

PHASE 4: PACKET

In this phase, we define a packet and then test it whether it is generating as expected. Packets are modelled using class. Packet class should be able to generate all possible packets randomly. Packet class variables and constraints have been derived from stimulus generation plan.

PHASE 5: MAILBOX

In this phase, we will write a driver and then instantiate the driver in environment and send the packet to the DUT. Driver is a class which generates the packets and then drives it to DUT input interface and pushes the packet into the mailbox.

PHASE 6: RECEIVER

In this phase, we will write a receiver and use the receiver in environment class to collect the packets coming from the output interface. The receiver collects the data bytes from the interface signals and then unpacks the bytes into packet and pushes them into mailbox.

PHASE 7: SCOREBOARD

The scoreboard has 2 mailboxes. One is used for getting packets from the driver and the other from the receiver. Then the packets are compared and if they don't match, an error is asserted.

PHASE 8: TESTCASES

We write implicit test bench 35 test cases and environment has 25 test cases using constraints for random test cases.

Verification:

There are 3 stages of verification.

1. *Testbench* – Based on this we are driving our input signals into the testbench considering most of the test cases. We simulate this testbench on QuestaSim
2. *Environment* – class based environment, it has randomization, constraint, etc. In the first process, we are generating a sequence. We have a packet class, then we have a generator class, which generates the sequence. Then we have driver class which drives the values to the DUT. We also have the monitor class, scoreboard, test and top. In the top, we connect the DUT with the testbench. This is the top of the testbench.
3. *Emulation* – which will be discussed further.

EMULATION:

We have implemented "Xtended testbench" (TBx) mode of emulation for our project. Our RTL is runs on Veloce, whereas testbench runs on comodel server. DUT on the Emulator uses clock, whereas testbench has no timing control and runs as a program in zero emulation time. Testbench (untimed portion) is

referred as HVL and DUT with transactor is referred as HDL. Data transfer between HVL and HDL occurs at a higher abstraction level, 'Transaction Level Modeling'. Reduction in unnecessary communication between HVL and HDL is achieved using this mode. TLM is the abstraction level of communication which uses module, task, functions, FIFO, rather than pin level connectivity.

There are 5 ways of transactions between CoModel and CoSimulator. We have used Bus Functional Model (BFM) to achieve the transaction between testbench and DUT. Bus Functional Model provides protocol level access using SystemVerilog Interfaces.

In a normal testbench, we send out the clock and reset signals to the DUT. From the DUT, all the pin connections go to the testbench in return. This is how a normal testbench and DUT works. In TBX mode, a role of HDL and HVL comes into picture. HDL is the DUT and HVL is top level module in which we include all the interfaces. HVL consists of systemverilog assertions, interfaces, etc. Transaction between HVL and HDL happens through TLM (Transaction Level Modelling). In this, whatever is the input/output or the communication between HVL and HDL happens in the form of packets and modules. Here, we do not do the pin to pin connections. Data is transferred and received with the help of packets. To implement this, we need some kind of protocol. TBX is having 4 types of protocols: DPI, BFM, Virtual Interface and UVM/OVM. For our code, we are using synthesizable BFM (Bus Functional Module). BFM is the media used to transfer the packets. BFM is a kind of an FSM which is implemented.

Puresim mode: Before emulating the entire design, we simulate it first in the puresim mode. Puresim is simulation carried out on QuestaSim. In Veloce, we do not have direct visibility into the transactor. Thus debugging becomes difficult. The purpose of PureSim is to make our debugging easy.

In our code, we have written the HVL, HDL which is the DUT, top, config file, makefile, and interface. These are the files implemented for the TBX mode. Everything in the TBX mode is dependent on clock.

RESULTS:

Verification/Simulation:

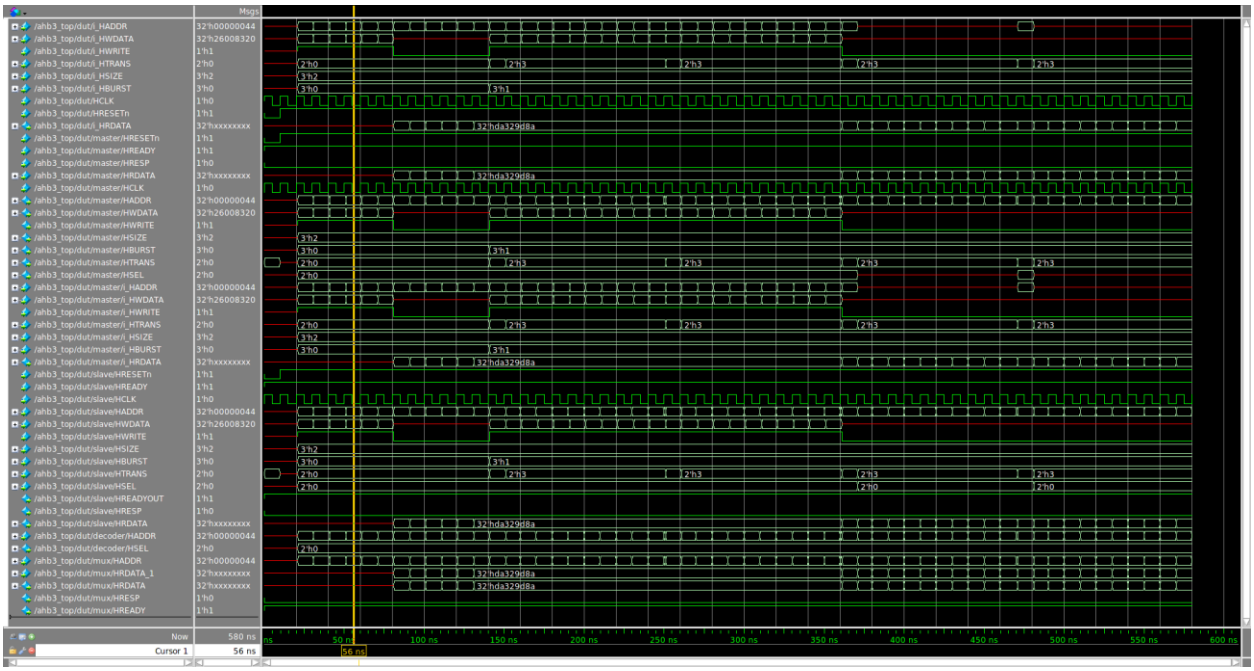
Transcript (snippet):

```
# vsim -c -novopt ahb3_top
# Start time: 17:20:37 on Mar 18,2018
# ** Warning: (vsim-8891) All optimizations are turned off because the -
novopt switch is in effect. This will cause your simulation to run very
slowly. If you are using this switch to preserve visibility for Debug or
PLI features please see the User's Manual section on Preserving Object
Visibility with vopt.
# //  Questa Sim-64
# //  Version 10.4c linux_x86_64 Jul 19 2015
# //
# //  Copyright 1991-2015 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
# //  WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# //  LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //  THIS DOCUMENT CONTAINS TRADE SECRETS AND COMMERCIAL OR FINANCIAL
# //  INFORMATION THAT ARE PRIVILEGED, CONFIDENTIAL, AND EXEMPT FROM
# //  DISCLOSURE UNDER THE FREEDOM OF INFORMATION ACT, 5 U.S.C. SECTION
552.
# //  FURTHERMORE, THIS INFORMATION IS PROHIBITED FROM DISCLOSURE UNDER
# //  THE TRADE SECRETS ACT, 18 U.S.C. SECTION 1905.
# //
# Loading sv_std.std
# Loading work.ahb3_top_sv_unit
# Loading work.ahb3_top
# Loading work.intf
# Loading work.test
# Loading work.top
# Loading work.ahb3lite_master
# Loading work.ahb3lite_slave
# Loading work.ahb3lite_decoder
# Loading work.ahb3lite_mux
# ** Warning: (vsim-3017) top.sv(39): [TFMPC] - Too few port connections.
Expected 20, found 19.
#    Time: 0 ns  Iteration: 0  Instance: /ahb3_top/dut/master File:
ahb3lite_master.sv
# ** Warning: (vsim-3722) top.sv(39): [TFMPC] - Missing connection for
port 'HSEL'.
```

```
#  driver function new
# scoreboard function
#                   20           0 ,, task write generator
#                   20  Enter into the driver
#                   20   enter monitor task
# scoreboard check...
# voloalaoaom
# HADDR from input packet     011
# HWDATA FROM input packet    6116fa6f
#  Data in the memory 00006116fa6f
# Data has been wriiten successfully
#
#                   30           1 ,, task write generator
#                   30  Enter into the driver
#                   30   enter monitor task
# scoreboard check...
# voloalaoaom
# HADDR from input packet     022
# HWDATA FROM input packet    10ed77d4
#  Data in the memory 000010ed77d4
# Data has been wriiten successfully
#
#                   40           2 ,, task write generator
#                   40  Enter into the driver
#                   40   enter monitor task
# scoreboard check...
# voloalaoaom
# HADDR from input packet     033
# HWDATA FROM input packet    32ccb7db
#  Data in the memory 000032ccb7db
# Data has been wriiten successfully
#
#                   50           3 ,, task write generator
#                   50  Enter into the driver
#                   50   enter monitor task
# scoreboard check...
# voloalaoaom
# HADDR from input packet     044
# HWDATA FROM input packet    26008320
#  Data in the memory 000026008320
# Data has been wriiten successfully
#
#                   60           4 ,, task write generator
#                   60  Enter into the driver
#                   60   enter monitor task
# scoreboard check...
# voloalaoaom
# HADDR from input packet     001
# HWDATA FROM input packet    37378d25
#  Data in the memory 000037378d25
# Data has been wriiten successfully
#
#                   70           5 ,, task write generator
#                   70  Enter into the driver
```

```
#                   70   enter monitor task
# scoreboard check...
# voloalaoaom
# HADDR from input packet     005
# HWDATA FROM input packet    da329d8a
#  Data in the memory 0000da329d8a
# Data has been wriiten successfully
#
#                   80   task read in generator
#                   80  Enter into the driver
#                   80   enter monitor task
# Scoreboard read
# hedlloo
# temp address = 011
# Read data from DUT 6116fa6f
# DATA from TB memory 00006116fa6f
# Data read successfully and it matches
#
#                   90   task read in generator
#                   90  Enter into the driver
#                   90   enter monitor task
# Scoreboard read
# hedlloo
```

Waveform:



Emulation:

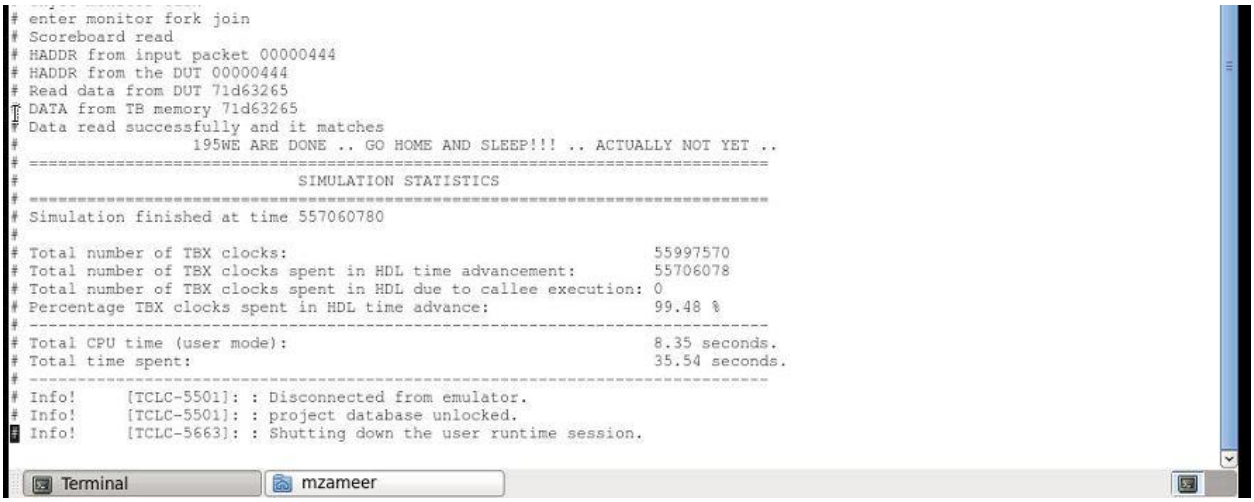
Transcript veloce (snippet):

```
# Scoreboard read
# temp address = 0ee
# Read data from DUT 407c1ac3
# DATA from TB memory 0000407c1ac3
# Data read successfully and it matches
#
#                  520     Read burst in generator loop
#                  520Enter into the driver
#                  520   enter monitor task
# Scoreboard read
# temp address = 0f2
# Read data from DUT 321b459b
# DATA from TB memory 0000321b459b
# Data read successfully and it matches
#
#                  530     Read burst in generator loop
#                  530Enter into the driver
#                  530   enter monitor task
# Scoreboard read
# temp address = 0f6
# Read data from DUT 0d7a45b7
# DATA from TB memory 00000d7a45b7
# Data read successfully and it matches
#
#                  540     Read burst in generator loop
#                  540Enter into the driver
#                  540   enter monitor task
# Scoreboard read
# temp address = 0fa
# Read data from DUT 9c6fad00
# DATA from TB memory 00009c6fad00
# Data read successfully and it matches
#
#                  550     Read burst in generator loop
#                  550Enter into the driver
#                  550   enter monitor task
# Scoreboard read
# temp address = 0fe
# Read data from DUT c4273ac1
# DATA from TB memory 0000c4273ac1
# Data read successfully and it matches
```

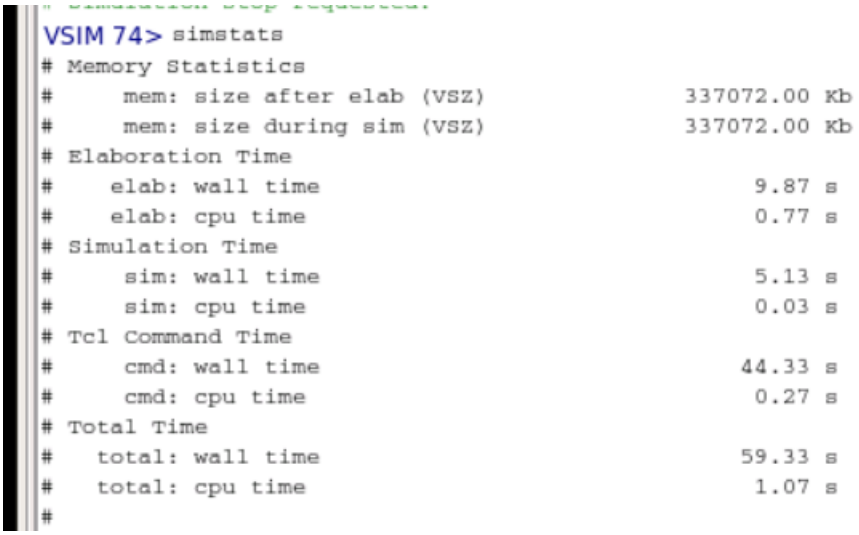
```
#
#                  560     Read burst in generator loop
#                  560Enter into the driver
#                  560   enter monitor task
# Scoreboard read
# temp address = 102
# Read data from DUT 9d1f43af
# DATA from TB memory 00009d1f43af
# Data read successfully and it matches
#
#                  570     Read burst in generator loop
#                  570Enter into the driver
#                  570   enter monitor task
# Scoreboard read
# temp address = 106
# Read data from DUT ebffe7cd
# DATA from TB memory 0000ebffe7cd
# Data read successfully and it matches
#
#                  580WE ARE DONE .. GO HOME AND SLEEP!!
# End time: 19:29:03 on Mar 18,2018, Elapsed time: 0:00:14
#
================================================================================
===
#                             SIMULATION STATISTICS
#
================================================================================
===
# Simulation finished at time 558162340
#
# Total number of TBX clocks:
56215151
# Total number of TBX clocks spent in HDL time advancement:
55816234
# Total number of TBX clocks spent in HDL due to callee execution: 0
# Percentage TBX clocks spent in HDL time advance:                 99.29 %
# ------------------------------------------------------------------------
-----
# Total CPU time (user mode):                                      8.30
seconds.
# Total time spent:                                                35.63
seconds.
# ------------------------------------------------------------------------
-----
# Info!     [TCLC-5501]: : Disconnected from emulator.
# Info!     [TCLC-5501]: : project database unlocked.
# Info!     [TCLC-5663]: : Shutting down the user runtime session.
```

Emulation vs Simulation Statistics:

Veloce Statistics:



Simulation Statistics:



Results Summary:

1. We are able to design Master and Slave and implemented basic functionality to write and read data using INCR, BURST mode of AHBlite protocol

2. Tested design using implicit TestBench

3. Class based verification environment was designed: packet, generator, driver monitor, interface,Environment, test, scoreboard, top were designed.

4. Results were verified using waveform, and $display on transcript(write and read succesfully)

5. TBX mode done successfully using Veloce Emulator

6. On Veloce, total time wall/CPU time reduced to nearly 55 percent as compared with the simulation

7. Percentage TBX Clocks spent in HDL is 99.48%

Future Scope of the Project:

- This code is parameterized to include more slaves in the future.
- We can also implement multiple masters and various data signals to send/read data to/from slaves.
- More test cases can be included to implement corner cases, checkers and Assertions.
- Can implement vacuous pass and fake pass.
- Optimization and reusability

Challenges faced:

- Designing DUT from scratch
- Designing class based verification
- Synchronization/Timing issue
- Major time was consumed in debugging
- Synthesizable testbench (we also tried on standalone mode)
- Deadloops, latches faced
- Had to improvise design accordingly

Bibliography:

1. www.testbench.in

2. AMBA AHBlite specifications from ARM

3. https://verificationacademy.com/

4. ECE 571 notes on interfaces, modports and Assertion

THANK YOU