ECE – 571

AMBA 3 AHB-LITE Protocol Design and Verification

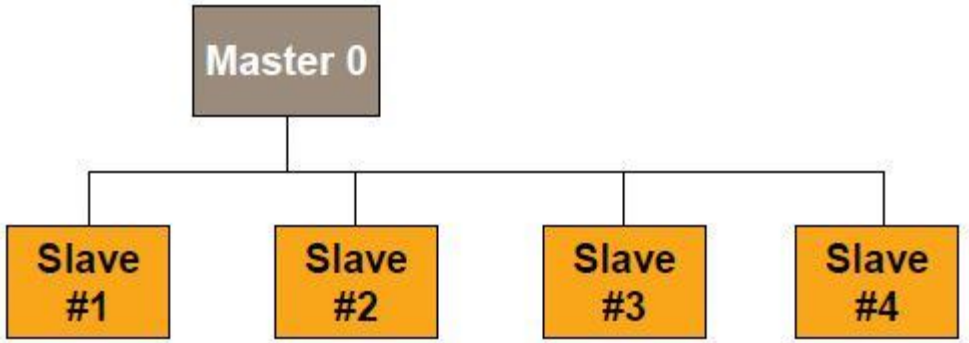
- Arjun Gopal
- Shrikrishna
- Nirlipth
- Udit

Main Objective

- Synthesizable Slave Memory module
- Bus Functional Model for the Master Side
- Usage of System Verilog constructs to perform functional Verification
- Perform Veloce emulation in TBX – BFM mode

Implemented both design and testbench from scratch

Introduction



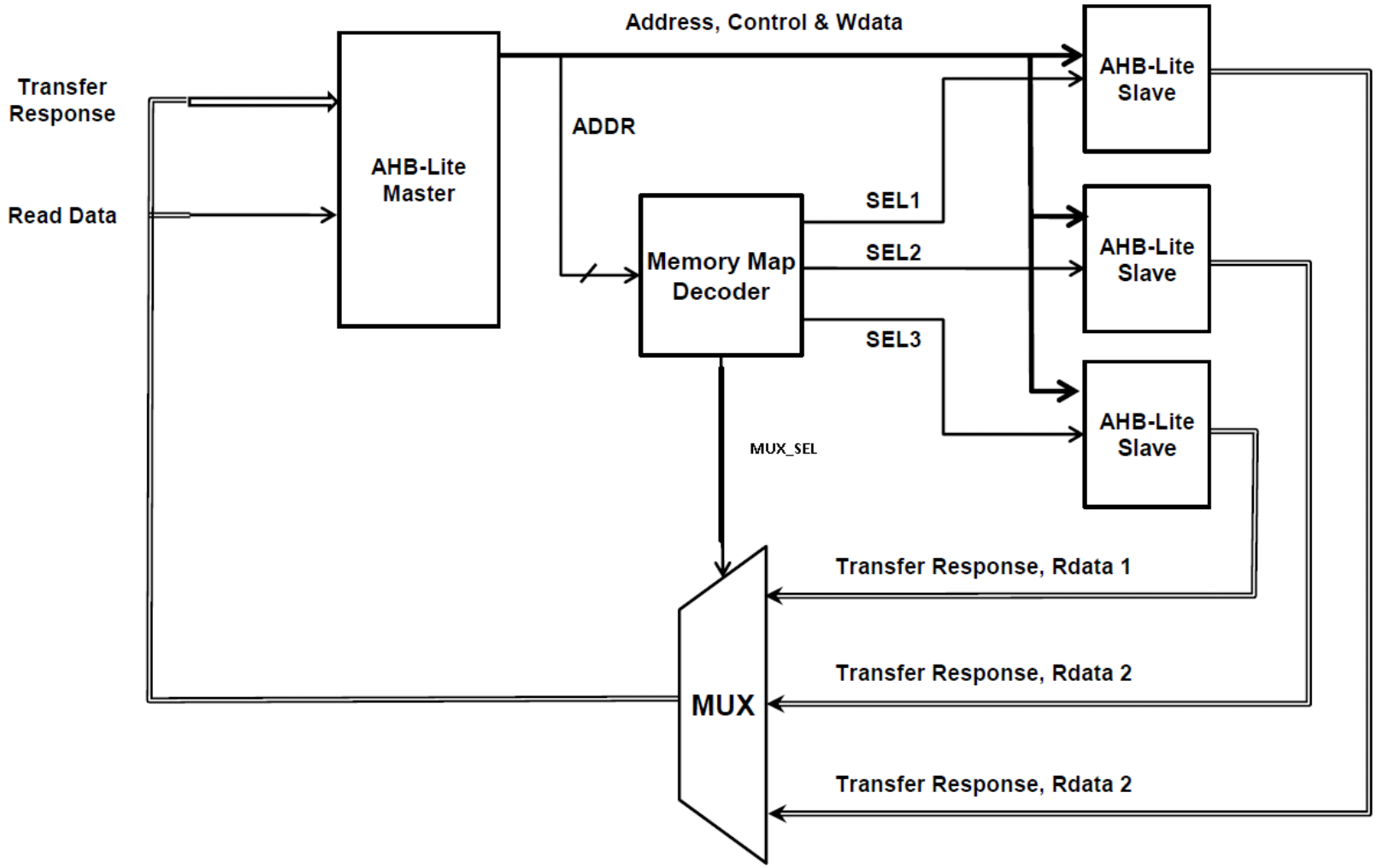
•Protocol developed by ARM
•AHB-Lite comes from AMBA- family of bus protocol specification
•Provide interconnects among all the functional blocks on a SoC

• Provides all the basic functions required by the majority of AMBA AHB slave and master design
• Also used in embedded processors with one or more CPUs or signal processors and multiple peripherals

•Single Master
•Multiple – Configurable Slaves
•Arbitration not required

* AHB → Advanced high performance bus
* AMBA → Advanced micro-controller bus architecture

Block diagram of the design



AHB-Lite Transactions

•Master Side:
- Basic Read
- Basic Write
- Burst Read – inc. burst lengths -1,4,8,16
- Burst Write – inc. burst lengths-1,4,8,16

Communication is initiated by master

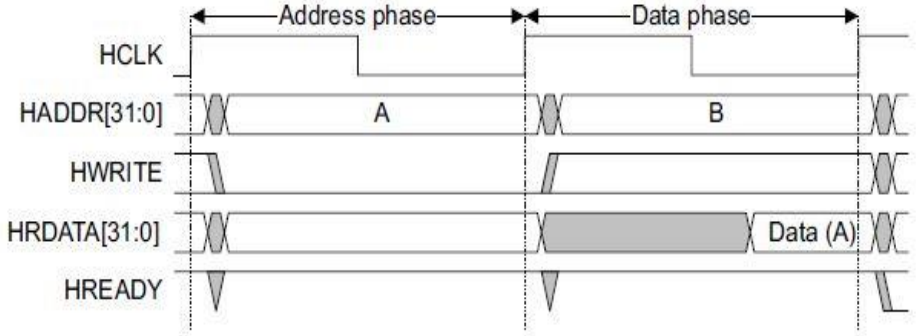


Figure 3-1 Read transfer

Slave Side:
- Can make the master wait
- Can give an Error Response
- But, slave can't terminate transaction; can ask master to insert wait state

•All transactions are pipelined

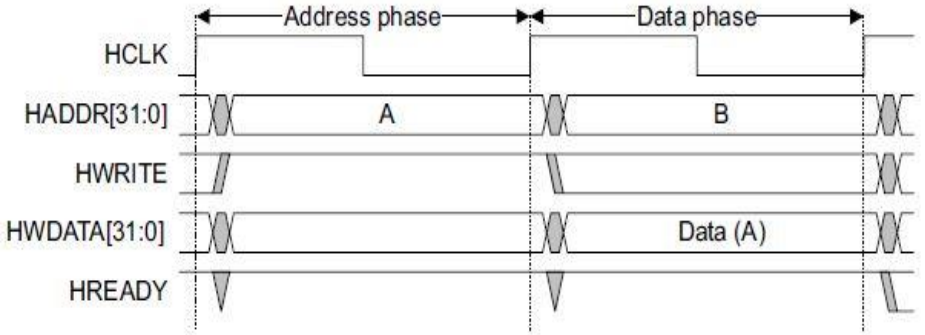


Figure 3-2 Write transfer

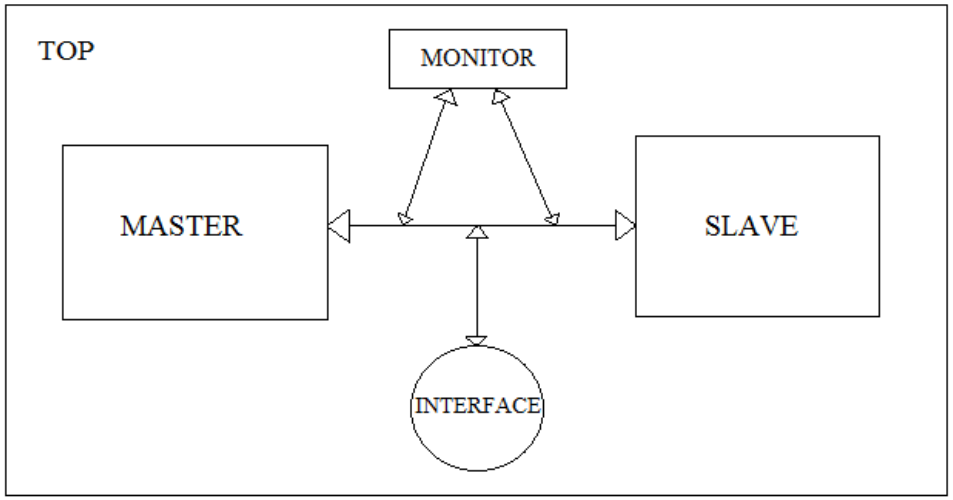
Environment

- Master : is written/programmed using a Bus Functional Model Approach (not Synthesizable)
  - It contains a host of tasks – read,write - that encompass the way the DUT is used.
  - By calling those tasks a read or a write would occur on the bus/interface of the DUT

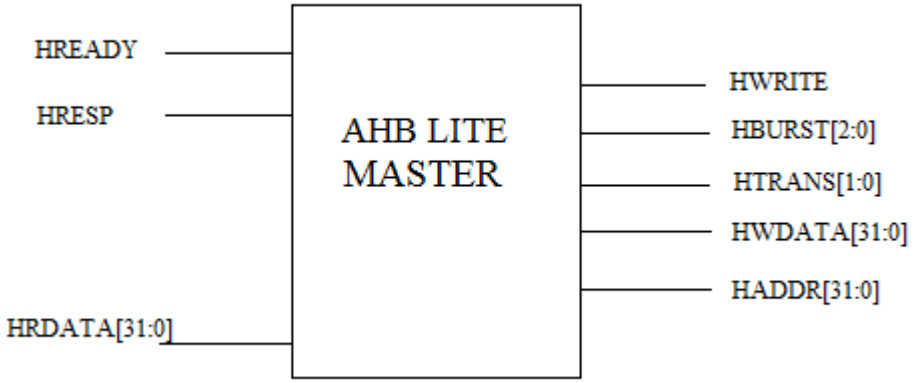
- Slave : is just a memory module which is our DUT and is programmed to be synthesizable. We have parameterized the number of slaves used in our module

- Monitor : This module just monitors the bus traffic and checks if the protocol is being followed by the ongoing transactions using assertions. The results have been updated in an Assertion Scoreboard.

- Interface : Consists of all the bus related signals and a slave modport to provide a proper lookup for the slave.



AHB Lite Master Signals



OUTPUTS

•HADDR : The 32 bit address based on which read and write happens.
•HWRITE : ‘1’ – write operation – ‘0’ – read operation.
•HBURST : Determines what type of burst is sent (4/8/16 Beats)
•HTRANS : Operates in 4 modes – Idle, busy(initiated by Master), Non-Sequential and Sequential.
•HWDATA : During write operation , data to be written is sent via this bus.

INPUTS

- HREADY : Slaves transfer response – ‘1’ previous transaction completed : ‘0’ wait state
- HRESP : Provides an error response if you try to write to a read only memory.
- HRDATA : Data which is read is transmitted through this bus.

SV constructs used:

- Constraint Randomization + Packet Class
- Packages – To define all parameters and global constants.
- Interface – Contains Master BFM and Bus Signals
- Program Block : Is scheduled in the Reactive region which prevents race condition from occurring
- Assertions – Used in the monitor module to verify the functionality of Protocol

```
//////////////PACKET CLASS//////////////////
class PacketClass;
    rand bit [31:0] Address_rand,Data_rand;
    constraint c {  Address_rand >  32'h00000000;
                    Address_rand    <  32'h00000400;
    }
endclass

//////////////PROGRAM BLOCK//////////////////
program test(AHBInterface InterfaceInstance);
    int rd,rd_wait;
    PacketClass pktCls;
//////////////INITIAL BLOCK//////////////////
    initial begin
        pktCls = new();
        assert(pktCls.randomize());
        //InterfaceInstance.write (32'h00000004, '1);
        InterfaceInstance.write(pktCls.Address_rand,pktCls.Data_rand);
        #20;
        //InterfaceInstance.read (32'h00000004, rd);
        InterfaceInstance.read (pktCls.Address_rand, rd);
```

```
interface AHBInterface(input HCLK,input HRESETn);
////////////////////////////////////////////////
    logic [ADDRESS_WIDTH-1:0]   HADDR;
    logic                       HWRITE;
    logic [HSIZE_WIDTH-1:0]     HSIZE;
    logic [BURST_SIZE-1:0]      HBURST;
    logic [TRANSFER_TYPE-1:0]   HTRANS;
    logic [DATA_WIDTH-1:0]      HWDATA;
    logic [DATA_WIDTH-1:0]      HRDATA;
    logic                       HREADY;
    logic                       HRESP;
    bit [ADDRESS_WIDTH-1:0]     ADDR;
    int i,j,k,n;
    int count=0;

modport Slave(  output HREADY,
                output HRESP,
                output HRDATA,
                input HADDR,
                input HWRITE,
                input HSIZE,
                input HBURST,
                input HTRANS,
                input HWDATA,
                input HCLK,
                input HRESETn
                    );
```

```
package definesPkg;
/*****************SLAVE******************/
parameter   ADDRESS_DEPTH   = 1024;
parameter   READ_ONLY_START_ADDRESS = 32'h00000000;
parameter   READ_ONLY_END_ADDRESS   = 32'h00000003;
parameter   WAIT_ADDRESS    = 32'h00000005;
/**************INTERFACE****************/
parameter ADDRESS_WIDTH = 32;
parameter DATA_WIDTH    = 32;
parameter HSIZE_WIDTH   = 3;
parameter BURST_SIZE    = 3;
parameter TRANSFER_TYPE = 2;
parameter IDLE          = 2'b00;
parameter BUSY          = 2'b01;
parameter NON_SEQ       = 2'b10;
parameter SEQ           = 2'b11;
endpackage
```

BFM Tasks

Read Task

```
task read(input bit [ADDRESS_WIDTH-1:0] addr,
      output logic [DATA_WIDTH-1:0] data);
    @(posedge HCLK);
    HADDR   = addr;
    HWRITE  = '0;
    HSIZE   = 3'b010;
    HBURST  = 3'b000;
    if (HRESP)      $display (" ERROR\n ");
    while (!HREADY);
    @(posedge HCLK);
    @(posedge HCLK);
    data = HRDATA;
endtask
```

Write Task

```
task write(input bit[ADDRESS_WIDTH-1:0] addr,
          input logic[DATA_WIDTH-1:0] data);
    @(posedge HCLK);
    HADDR   = addr;
    HWRITE  = '1;
    HSIZE   = 3'b010;
    HBURST  = '0;
    while (!HREADY) wait(HREADY);
    @(posedge HCLK);
    HWDATA  = data;
    @(posedge HCLK);
    if (HRESP)      $display ("ERROR\n");
endtask
```

Burst Read

```
task burst_read (bit [ADDRESS_WIDTH-1:0] addr,
                 input int BEATS,
                 input int busy,
                 output bit [DATA_WIDTH-1:0] data_burst[31:0]);
    automatic int i,j=0;
    @(posedge HCLK);
    HADDR = addr;
    HTRANS = NON_SEQ;
    HWRITE = 1'b0;
    @(posedge HCLK);
    HADDR = addr +1;
    repeat(busy)
        begin
        j++;
        HTRANS = BUSY;
        @(posedge HCLK);
        if(j == busy) HTRANS = SEQ;
        end
    i=0;
    repeat(BEATS-2)
        begin
        @(posedge HCLK);
        HADDR = HADDR + 1;
        HTRANS = SEQ;
        data_burst4[i] = HRDATA;
        i=i+1;
        end
        @(posedge HCLK);
        data_burst4[i] = HRDATA;
        i=i+1;
        @(posedge HCLK);
        data_burst4[i] = HRDATA;
endtask
```

Burst Write

```
fork
    begin
    repeat (BEATS)
        begin
        if (i==0)           HTRANS = NON_SEQ;
        HADDR = addr;
        HWRITE  = 1;
        @(posedge HCLK);
        HADDR = addr;
        while (!HREADY) wait (HREADY);
        addr = addr+1;
        i=i+1;
        if (busy ==32'd1)
            begin
            if (i==2)
                begin
                HTRANS = BUSY;
                @(posedge HCLK);
                end
            end

        if (busy == 32'd2)
            begin
            if (i==3)
                begin
                HTRANS = BUSY;
                @(posedge HCLK);
                @(posedge HCLK);
                end
            end

        if ( (busy==32'd1)||(busy ==32'd2) )
            begin
            if ((i!=2) || (i!=3))
                HTRANS = SEQ;
            end
            else
                HTRANS = SEQ;
```

Burst Write Cntd.

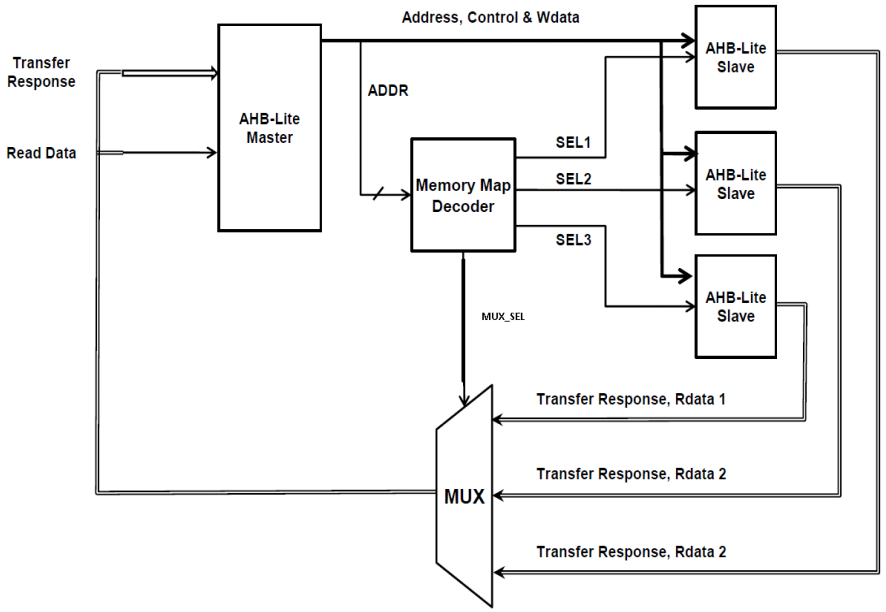
```
        begin
        repeat (BEATS)
            begin
            @(posedge HCLK);
            HWDATA = data4[j];
            while (!HREADY) wait(HREADY);
            j=j+1;
            k = k+1;
            if (busy ==32'd1)
                begin
                if (k==2)
                    begin
                    @(posedge HCLK);
                    end
                end

            if (busy == 32'd2)
                begin
                if (k==3)
                    begin
                    @(posedge HCLK);
                    @(posedge HCLK);
                    end
                end
            end
        end
    join
endtask
```

AHB Lite Slave Module

AHB Slave Top Module encloses

- Configurable No. of Slaves implemented with generate – endgenerate block
- Parameterized Decoder
- Parameterized Mux
- Default Slave (sends error responses if address given is out of the address space)




```
genvar i;
    generate
        for (i=0;i<`NoOfSlaves;i++) begin
            AHBSlaveMemory AHBMemi (.HCLK(SlaveInterface.HCLK),
                                    .HRESETn(SlaveInterface.HRESETn),
                                    .HADDR(SlaveInterface.HADDR[AddrSpace-1:0]),
                                    .HWDATA(SlaveInterface.HWDATA),
                                    .HTRANS(SlaveInterface.HTRANS),
                                    .HWRITE(SlaveInterface.HWRITE),
                                    .HSEL(HSEL[i]),
                                    .wait_slave_to_master(wait_slave_to_master),
                                    .HRDATA(HRDATA_BUS[i]),
                                    .HRESP(HRESP_BUS[i]),
                                    .HREADY(HREADY_BUS[i]));
        end
    endgenerate
```

Parameterized decoder and Mux

```
//paramterized decoder
module decoder(input logic [`NoOfSlaves-1:0] Decode_address,output logic [`NoOfSlaves-1:0] HSEL );
    assign HSEL = (`NoOfSlaves'b01) << Decode_address;
endmodule
```

```
//paramterized mux
assign decode_address = SlaveInterface.HADDR[AddrSpace+$clog2(`NoOfSlaves)-1:AddrSpace];
assign SlaveInterface.HRESP  = HRESP_BUS[decode_address];
assign SlaveInterface.HREADY = HREADY_BUS[decode_address];
assign SlaveInterface.HRDATA = HRDATA_BUS[decode_address];
```

Slave responses

- HREADY = 0 Slave asks master to insert wait states
- HREADY = 1, HRESP = 0 Successful transfer response (OK response)
- HREADY = 1, HRESP = 1 Error response

Error responses

- If address exceeds the address space, default slave provides an error response
- If the master tries to write into a read-only address, an error response is generated

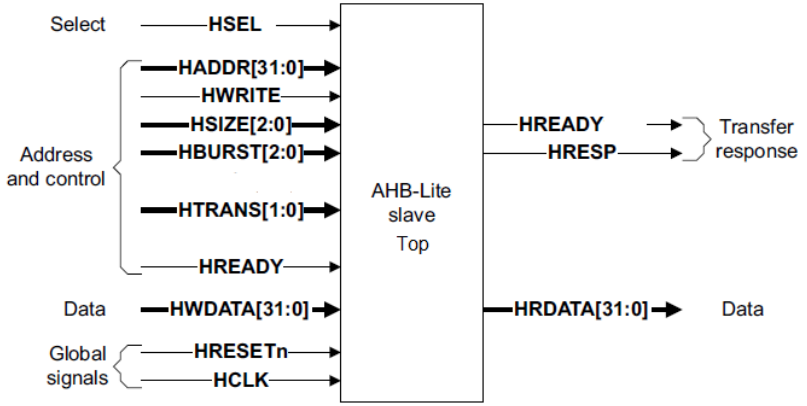
Slave Memory

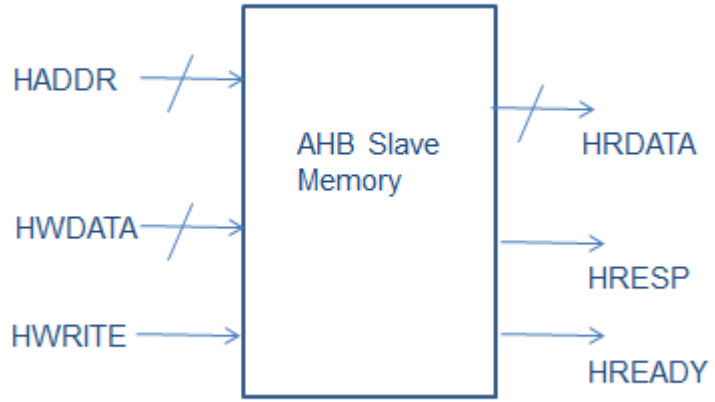
•A RAM Block used as Slave.

•Read and Write operations performed with a read / write enable signal

•RAM specs:

- •No. of Slaves Configurable between 1 and 4
- •Width Configurable at compile time
- •Configurable address space of a slave between 000H and 3FFH
- •The addresses continue for every next slave. i.e the 2nd slave has addresses from 400H to 7FFH and so on.
- •00H to 03H addresses in every RAM are configured to be read-only addresses





Verification plan

- Assertions

. Usage of Concurrent and Immediate Assertions

```
# ----------------------------------------------------------------------------------------------------
# ------------------------------------ASSERTION SCOREBOARD-------------------------------------------
# ****************************************************************************************************
# TYPE OF CHECK                                TOTAL COUNT            PASS COUNT              FAIL COUNT
# ----------------------------------------------------------------------------------------------------
#  error_check                                  10                     10                      0
#  read_only_error_check                         6                      6                      0
#  basic_write_error                            34                     34                      0
#  basic_read_error_check                       20                     20                      0
#  basic_burst_write_check                      32                     32                      0
#  basic_burst_read_check                       36                     36                      0
#  HREADY_check                                 12                     12                      0
#  busy_to_sequential_check                     35                     35                      0
#  Sequential_wait_check                        11                     11                      0
#  bursts_count_check4                          10                     10                      0
#  bursts_count_check8                          10                     10                      0
#  bursts_count_check16                         12                     12                      0
#  address_change4                              13                     13                      0
#  address_change8                              25                     25                      0
#  address_change16                             24                     24                      0
# End time: 15:19:20 on Dec 03,2015, Elapsed time: 0:00:06
# Errors: 0, Warnings: 0
```

Verification plan

```
# ------------------------------------------------------------------------------------------
# ----------------------------------------ASSERTION SCOREBOARD----------------------------------------
# ******************************************************************************************
# TYPE OF CHECK                                TOTAL COUNT             PASS COUNT              FAIL COUNT
# ------------------------------------------------------------------------------------------
#  error_check                                  96                      15                      81
#  read_only_error_check                        11                       9                       2
#  basic_write_error                            23                      23                       0
#  basic_read_error_check                       20                      20                       0
#  basic_burst_write_check                      21                      21                       0
#  basic_burst_read_check                       36                      36                       0
#  HREADY_check                                 12                      12                       0
#  busy_to_sequential_check                     19                      19                       0
#  Sequential_wait_check                       122                     111                      11
#  bursts_count_check4                          10                      10                       0
#  bursts_count_check8                          10                      10                       0
#  bursts_count_check16                         12                      12                       0
#  address_change4                              13                      13                       0
#  address_change8                              25                      25                       0
#  address_change16                             24                      24                       0
# End time: 15:57:08 on Dec 03,2015, Elapsed time: 0:00:05
# Errors: 0, Warnings: 0
```

Verification plan

```
property basic_read;
    //@(posedge Bus.HCLK) disable iff ((Bus.HTRANS==2'b00 || Bus.HTRANS==2'b01) && Bus.HBURST!='0) Bus.HWRITE ##1 Bus.HREADY;
    @(posedge Bus.HCLK) disable iff ((Bus.HTRANS==2'b00 || Bus.HTRANS==2'b01) && Bus.HBURST>'0 || (Bus.HADDR > ((2**10)* `NoOfSlaves))) $fell(Bus.HWRITE) |=> Bus.HREADY;
    //assertion worked - for basic read -> hwrite is detected low at the 2nd clk,at the third clk,hready goes high
endproperty

// assertion worked - basic burst write
property basic_burst_write;
@(posedge Bus.HCLK)
disable iff (Bus.HTRANS == 2'b01)
((Bus.HWRITE==1)&&(Bus.HTRANS == 2'b10) )|=>    (Bus.HREADY=='1) ;
endproperty
```

```
property seq_check;
@(posedge Bus.HCLK)
(( (Bus.HWRITE=='1) || (Bus.HWRITE=='0) ) && (Bus.HTRANS==2'b10) ) |=> (Bus.HTRANS == 2'b11);
endproperty
```

```
property HREADY_check;
@(posedge Bus.HCLK) (Bus.HREADY == 1'b0) |=> $stable (Bus.HADDR && Bus.HWRITE && Bus.HWDATA) ;
endproperty
```

Verification plan

Test Cases

```
//////////////////////////////////////BASIC 4 BEAT BURST READ AND WRITE - SLAVE 2////////////////////////////////////////////////?/////

   InterfaceInstance.burst_write (32'h0700,4, 0,data_burst);

   #20;
   InterfaceInstance.burst_read (32'h0700,4, 2,data_burst_read);
   #20;
   $display ("DATA - 2 BUSY - 4 BEAT BURST - SLAVE 2  = %d, %d, %d, %d\n", data_burst_read[0], data_burst_read[1], data_burst_read[2], data_burst_read[3] );

   //InterfaceInstance.read (32'h00100706, rd);
   //$display ("DATA @ address 04h = %h", rd_wait);


   ////////////////////////////////BASIC 8 BEAT BEAT BURST READ AND WRITE - SLAVE 1/////////////////////////////////////////////

   rfaceInstance.burst_write (32'h0100,8, 0,data_burst);


   rfaceInstance.burst_read (32'h0100,8, 0,data_burst_read);

   play ("DATA - 1 BUSY - 8 BEAT BURST - SLAVE 1 = %d, %d, %d, %d, %d, %d, %d, %d\n", data_burst_read[0], data_burst_read[1], data_burst_read[2], data_burst_read[3] , data_burst_read[4], data_burst_read[5], data_burst_read[6], data_burst_read[7] );
```

Emulation -veloce

- Emulation on veloce solo performed in TBX-BFM Mode
- Design and environment partitioned into top_tb and top_hdl files
- Used pragmas and clocked tasks to get the BFM running on Veloce Solo along with Synthesizable Slave Top module
- top_tb is the testbench that resides on Veloce Solo Host server. Constrained randomization used to generate address particularly for Slave 1 & Slave 2
- Tests driven from Program endprogram block in top_tb

Supported features of the Protocol

- Basic Read
- Basic Write
- Burst Read for Burst length - 4
- Burst Write for Burst length - 4
- Busy states and wait states not supported- hard to implement after partitioning

top_hdl : Interface, clocked tasks in the BFM
generate –endgenerate block to generate configurable no. of Synthesizable Slaves
top_tb: Class, Program, final, tasks

Pragmas used:
tbx clkgen
pragma tbx xtf
pragma attribute top_hdl partition_module_xrtl
pragma attribute AHBInterface partition_interface_xif

Emulation Results - PureSim

```
----------------------
First Slave operations
----------------------
Basic operations:


Address:000003ed       DataWritten:39944c90    Data Read:39944c90 ----- PASS
Address:00000257       DataWritten:684b2ef0    Data Read:684b2ef0 ----- PASS
Address:0000003d       DataWritten:36c11d55    Data Read:36c11d55 ----- PASS
Address:000001b4       DataWritten:9e39f4a3    Data Read:9e39f4a3 ----- PASS
Address:000003a2       DataWritten:d923ad20    Data Read:d923ad20 ----- PASS
Address:000000b7       DataWritten:2856744e    Data Read:2856744e ----- PASS
Address:000003e9       DataWritten:649a607d    Data Read:649a607d ----- PASS
Address:000002bd       DataWritten:b5a9a03b    Data Read:b5a9a03b ----- PASS
Address:00000257       DataWritten:e1da356c    Data Read:e1da356c ----- PASS
Address:000002b1       DataWritten:afd15eef    Data Read:afd15eef ----- PASS
----------------------
First Slave Burst operations
----------------------
-----------------
PASS
Address:00000006       Data written:0000003b   Data Read:0000003b
Address:00000007       Data written:0000003d   Data Read:0000003d
Address:00000008       Data written:0000003f   Data Read:0000003f
Address:00000009       Data written:00000041   Data Read:00000041
--------------------------
```

```
--------------------------
Second Slave operations
--------------------------
Address:000005bd       DataWritten:6f424c03    Data Read:6f424c03 ----- PASS
Address:00000692       DataWritten:6281acc4    Data Read:6281acc4 ----- PASS
Address:00000447       DataWritten:bbac35c3    Data Read:bbac35c3 ----- PASS
Address:00000661       DataWritten:d5579d1b    Data Read:d5579d1b ----- PASS
Address:000005cf       DataWritten:524ddfb9    Data Read:524ddfb9 ----- PASS
Address:00000787       DataWritten:975e6523    Data Read:975e6523 ----- PASS
Address:00000468       DataWritten:93e2db7f    Data Read:93e2db7f ----- PASS
Address:0000058d       DataWritten:0f4f4764    Data Read:0f4f4764 ----- PASS
Address:000007a1       DataWritten:af224f5b    Data Read:af224f5b ----- PASS
Address:00000748       DataWritten:713f883f    Data Read:713f883f ----- PASS
---------------------
 Burst operations on Slave 2
-----------------
PASS
Address:00000706       Data written:0000003b   Data Read:0000003b
Address:00000707       Data written:0000003d   Data Read:0000003d
Address:00000708       Data written:0000003f   Data Read:0000003f
Address:00000709       Data written:00000041   Data Read:00000041
--------------------------
```

```
**************END OF SIMULATION***********

Basic Reads on Slave 1:                 10
Basic Writes on Slave 1:                10
Basic Reads on Slave 2:                 10
Basic Writes on Slave 2:                10
Pass count:                             20
Fail count:                              0
---------------------------------------
Burst Reads on Slave 1:                  1
Burst Writes on Slave 1:                 1
Burst Reads on Slave 2:                  1
Burst Writes on Slave 2:                 1
Pass count:                              2
Fail count:                              0
```

Emulation results - Veloce

```
----------------------
First Slave operations
----------------------
Basic operations:


Address:000003ed       DataWritten:39944c90    Data Read:39944c90 ----- PASS
Address:00000257       DataWritten:684b2ef0    Data Read:684b2ef0 ----- PASS
Address:0000003d       DataWritten:36c11d55    Data Read:36c11d55 ----- PASS
Address:000001b4       DataWritten:9e39f4a3    Data Read:9e39f4a3 ----- PASS
Address:000003a2       DataWritten:d923ad20    Data Read:d923ad20 ----- PASS
Address:000000b7       DataWritten:2856744e    Data Read:2856744e ----- PASS
Address:000003e9       DataWritten:649a607d    Data Read:649a607d ----- PASS
Address:000002bd       DataWritten:b5a9a03b    Data Read:b5a9a03b ----- PASS
Address:00000257       DataWritten:e1da356c    Data Read:e1da356c ----- PASS
Address:000002b1       DataWritten:afd15eef    Data Read:afd15eef ----- PASS
----------------------
First Slave Burst operations
----------------------
-----------------
FAIL
Address:00000006       Data written:0000003b   Data Read:0000003d
Address:00000007       Data written:0000003d   Data Read:0000003f
Address:00000008       Data written:0000003f   Data Read:00000041
Address:00000009       Data written:00000041   Data Read:0000003b
-------------------------
```

```
-------------------------
Second Slave operations
-------------------------
Address:000005bd       DataWritten:6f424c03    Data Read:6f424c03 ----- PASS
Address:00000692       DataWritten:6281acc4    Data Read:6281acc4 ----- PASS
Address:00000447       DataWritten:bbac35c3    Data Read:bbac35c3 ----- PASS
Address:00000661       DataWritten:d5579d1b    Data Read:d5579d1b ----- PASS
Address:000005cf       DataWritten:524ddfb9    Data Read:524ddfb9 ----- PASS
Address:00000787       DataWritten:975e6523    Data Read:975e6523 ----- PASS
Address:00000468       DataWritten:93e2db7f    Data Read:93e2db7f ----- PASS
Address:0000058d       DataWritten:0f4f4764    Data Read:0f4f4764 ----- PASS
Address:000007a1       DataWritten:af224f5b    Data Read:af224f5b ----- PASS
Address:00000748       DataWritten:713f883f    Data Read:713f883f ----- PASS
--------------------
 Burst operations on Slave 2
-----------------
FAIL
Address:00000706       Data written:0000003b   Data Read:0000003d
Address:00000707       Data written:0000003d   Data Read:0000003f
Address:00000708       Data written:0000003f   Data Read:00000041
Address:00000709       Data written:00000041   Data Read:0000003b
-------------------------
```

```
**************END OF SIMULATION**********

Basic Reads on Slave 1:                 10
Basic Writes on Slave 1:                10
Basic Reads on Slave 2:                 10
Basic Writes on Slave 2:                10
Pass count:                             20
Fail count:                              0
---------------------------------------
Burst Reads on Slave 1:                  1
Burst Writes on Slave 1:                 1
Burst Reads on Slave 2:                  1
Burst Writes on Slave 2:                 1
Pass count:                              0
Fail count:                              2
```

Challenges Faced

- Figuring out the testing environment.
- Design from Scratch
- Fork and Join issues
- Veloce – To make the BFM synthesizable, to infer memory

Work distribution

Nirlipth – BFM design, Testcases
Arjun – Assertions
Udit – Slave , Testing
Shrikrishna – Slave configurable, Emulation