EN4020 - Advanced Digital Systems

System Bus Design

Department of Electronic & Telecommunication Engineering
University of Moratuwa.

CONTENTS

TABLE OF FIGURES

Signal	From	To	Task
clk		Bus	This clock times all bus transfers. All signal timings are related to the rising edge of HCLK
sb_lock_m1	Master 1	Arbiter	Lock request
req1	Master 1	Arbiter	Request for bus
req2	Master 2	Arbiter	Request for bus
Sb_lock_m2	Master 2	Arbiter	Lock request
resp0	Slave 0	Mux3-1_Resp	Response from slave 0
resp1	Slave 1	Mux3-1_Resp	Response from slave 1
resp2	Slave 2	Mux3-1_Resp	Response from slave 2
Resp	Mux3-1_Resp		
HADDR_M1 [14:0]	Master 1	ADD_mux (This is a 2 to 1 mux for address & control)	Address from master 1
HADDR_M2 [14:0]	Master 2	ADD_mux	Address from master 2
HADDR [14:0]	ADD_mux	Decoder	For slave selection first 2 bits specify the slave 00 - slave 0 01 - slave 1 10 - slave 2
RDATA_S0 [31:0]	Slave 0	Read mux (This is a 3-1 mux for read data)	Getting read data from slave 0
RDATA_S1 [31:0]	Slave 1	Read mux	Getting read data from slave 1

RDATA_S2 [31:0]	Slave 2	Read mux	Getting read data from slave 2
WDATA_M1 [31:0]	Master 1	Write data mux	Getting write data from master 1
WDATA_M2 [31:0]	Master 2	Write data mux	Getting write data from master 1
Sb_split_ar [1:0]	Slave	Arbiter	
sel_slave [1:0]	Decoder	Mux3-1_Resp Read mux	Sending slave selection data to mux
WDATA [31:0]	Write data mux	Slave	Sending write data to slave
gnt1	Arbiter	Master 1	Grant the master 1 (master 1 - granted master 2 - not granted)
gnt2	Arbiter	Master 2	Grant the master 2. (master 1 - not granted master 2 - granted)
Sb_masters [1:0]	Arbiter	ADD_mux (Address mux)	Specify the selected master. 00 - no master 01 - master 1 10 - master 2
Sb_mastlock	Arbiter	Slave	1 - lock 0 - no lock
RDATA [31:0]	Read mux	Master	Data bus carries read data from mux to master
sel_0	Decoder	Slave 0	1 - selected 0 - not selected
sel_1	Decoder	Slave 1	1 - selected 0 - not selected
sel_2	Decoder	Slave 2	1 - selected 0 - not selected
sel_slave [1:0]	Decoder	Mux3-1_Resp Read mux	Sending slave selected data to muxes 00 - slave 0

			01 - slave 1 10 - slave 2
WDATA [31:0]	Write data mux	Slave	Sending write data to slave

1. Signal List

The following table gives an insight to the signals that we have used in our bus architecture.

2. Top Module Verification

In our design we created our bus using Verilog in two stages. First we created top module using 3 mux and a decoder. Then we created our bus by instantiating the top module and the arbiter.

2.1. Bus Architecture Diagram

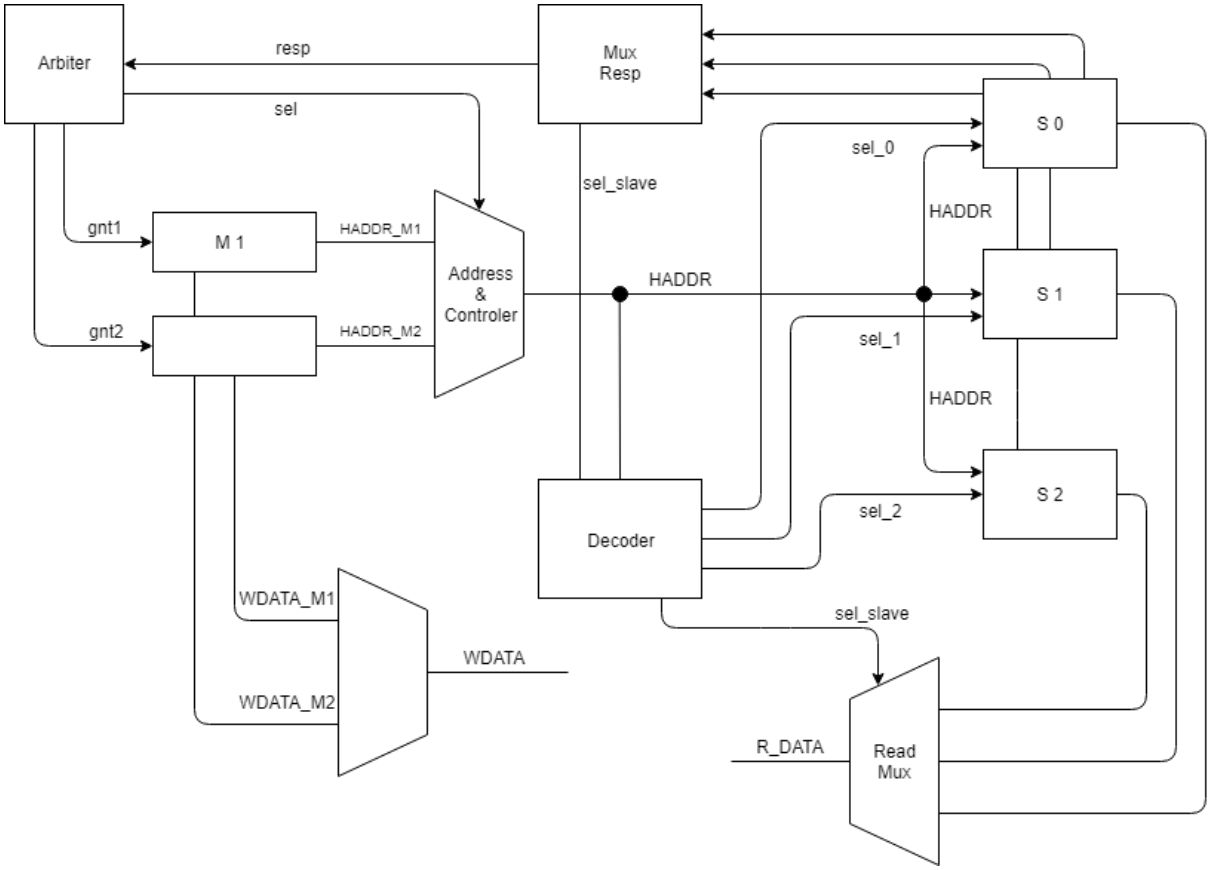
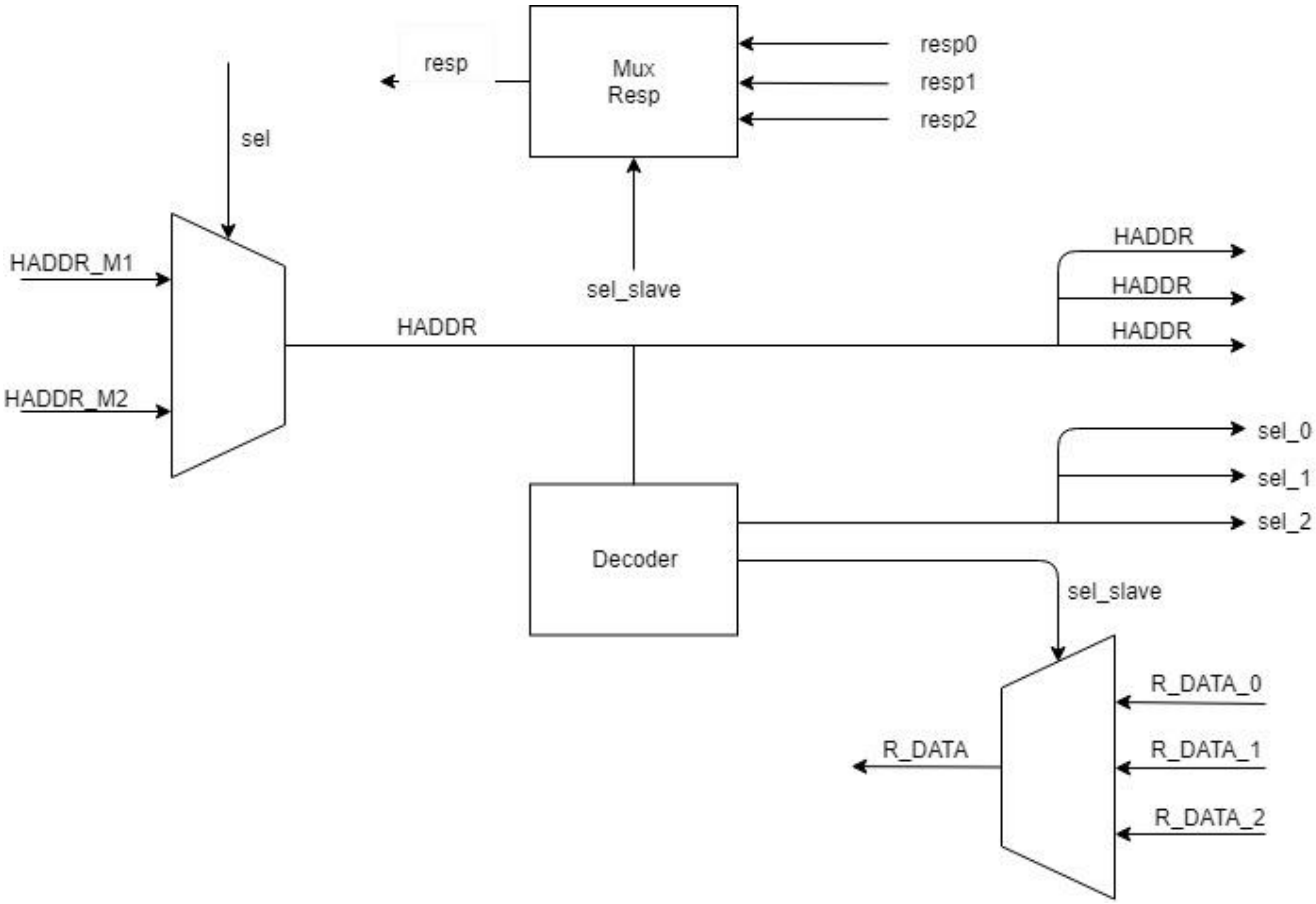


Figure 2-1 – Bus architecture diagram

2.2. Top Module Architecture Diagram



2.3. Top Module Test Cases

We have simulated our top module corresponding to many input senarios. Within each input set is kept for 50 ns. And then switched to another input set.

- rst- Reset
- sb_lock_m1 - Master 1 request to lock the bus
- sb_lock_m2 - Master 2 request to lock the bus
- req1 - request to get the bus by master 1
- req2 - request to get the bus by master 2
- resp - Respond from slave to arbiter resp=3 means the Split
- sb_split_ar - This indicates the Splitx operation.

1. Test Case 1 - Reset Test(rst=1, sb_lock_m1=0, sb_lock_m2=0, req1=0, req2=1)

2. Test Case 2 - One master request(rst=1, sb_lock_m1=0, sb_lock_m2=0, req1=0, req2=1)

3. Test Case 3 - (rst=0, sb_lock_m1=0, sb_lock_m2=0, req1=0, req2=1)

4. Test Case 4 - (rst=0, sb_lock_m1=1, sb_lock_m2=0, req1=1, req2=0)

5. Test Case 5 - (rst=0, sb_lock_m1=1, sb_lock_m2=0, req1=0, req2=1)

6. Test Case 6 - (rst=0, sb_lock_m1=0, sb_lock_m2=0, req1=0, req2=1)

7. Test Case 7 - Two Master Request(rst=0, sb_lock_m1=0, sb_lock_m2=0, req1=1, req2=1)

8. Test Case 8 - (rst=0, sb_lock_m1=0, sb_lock_m2=0, req1=0, req2=1)

9. Test Case 9 - Split(rst=0, sb_lock_m1=0, sb_lock_m2=0, req1=0, req2=1,resp=3)

10. Test Case 10 - Splitx(rst=0, sb_lock_m1=0, sb_lock_m2=0, req1=0, req2=1,resp=3,sb_split_ar=2)

11. Test Case 11- Two Master Request(rst=0, sb_lock_m1=0, sb_lock_m2=0, req1=1, req2=1,resp=3,sb_split_ar=2)

Here Test Case 9 and 10 demonstrate the Split transaction scenario.

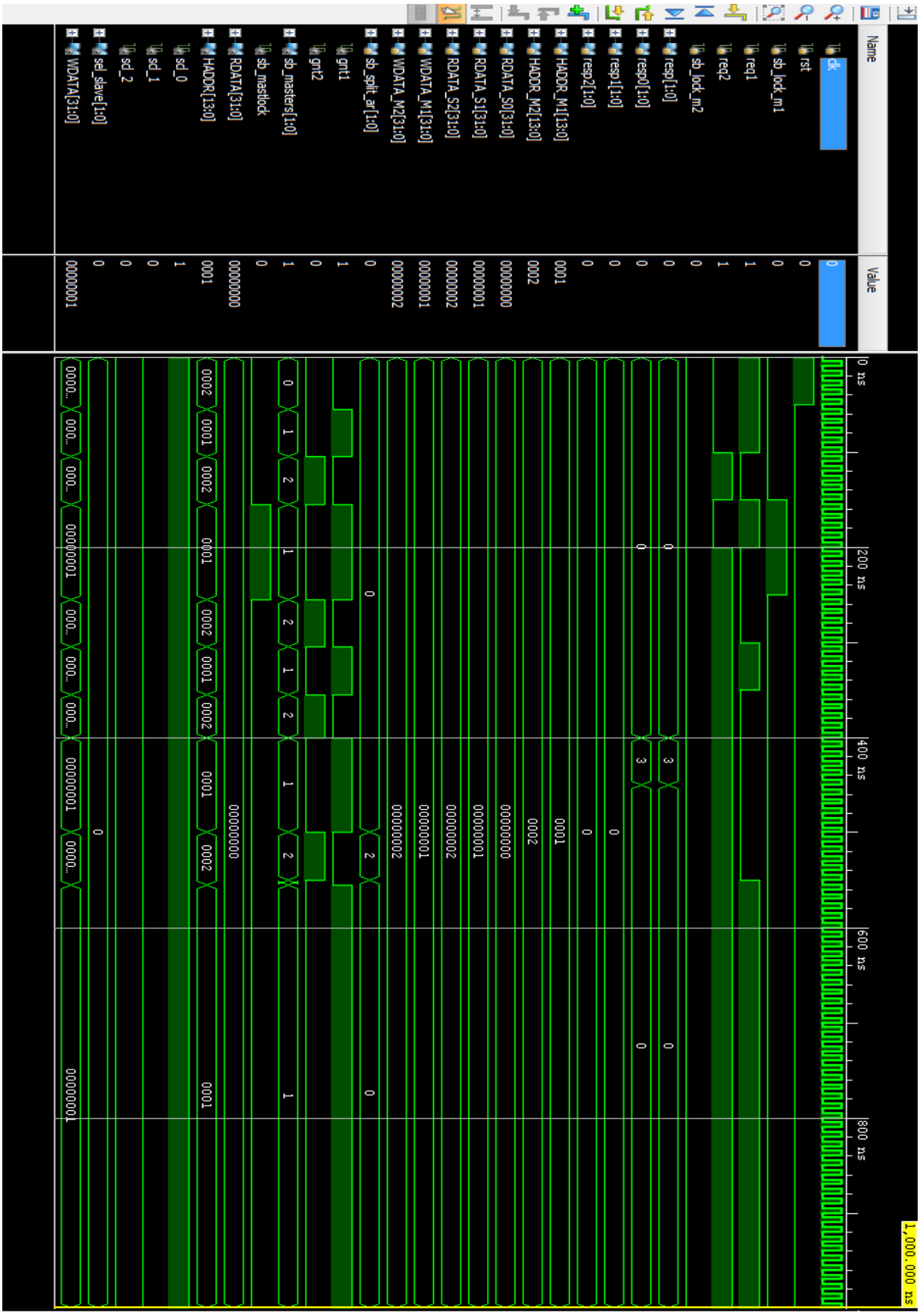


Figure 2-2 : Top module test cases simulation

2.4. Bus Module Source Code

```
module Bus(

input wire clk,
input wire rst,//reset
input wire sb_lock_m1,//lock request from m1 to arbiter
input wire req1,//request to aquire the bus from Master1
input wire req2,//request to aquire the bus from Master2
input wire sb_lock_m2,//lock request from m2 to arbiter
input wire [1:0] resp,//Responce from mux
input wire [1:0] resp0,//responce from slave 0
input wire [1:0] resp1,//responce from slave 1
input wire [1:0] resp2,//responce from slave 2
input wire [13:0] HADDR_M1,//Address of Master 1
input wire [13:0] HADDR_M2,//Address of Master 2
input wire [31:0] RDATA_S0,//Read Data from Slave 0
input wire [31:0] RDATA_S1,//Read Data from Slave 1
input wire [31:0] RDATA_S2,//Read data from Slave 2
input wire [31:0] WDATA_M1,//Write data from M1
input wire [31:0] WDATA_M2,//Write data from M2
input wire [1:0] sb_split_ar,//Splitx from slave to arbiter 01-GNT Master 1
// 10-GNT Mater 2

output wire gnt1,// Grant for Master 1
output wire gnt2,//Grnt for Master 2
output wire [1:0] sb_masters,//00 - No master // 01 - Master 1 // 10 - Master
2
output wire sb_mastlock,//output from arbiter about master lock 1- lock 0-
no lock to all slaves
output wire [31:0] RDATA,//Read data from slave mux
output wire [13:0] HADDR, // Address from Address mux
output wire sel_0,//input to slave 0 about its selection 1 -selected
output wire sel_1,
output wire sel_2,
output wire [1:0] sel_slave,// from Decoder about the slave selected
output wire [31:0] WDATA//Write data from write data mux

);


    Arbiter Arbiter_Instance(clk,
     rst,
     req1,
     sb_lock_m1,
     req2,
     sb_lock_m2,
     sb_split_ar,
     resp,
     gnt1,
     gnt2,
     sb_masters,
     sb_mastlock
     );

    top_module top_module_instance(
         resp0,
         resp1,
         resp2,
         HADDR_M1,
         HADDR_M2,
         RDATA_S0,
         RDATA_S1,
```

```
        sb_masters,
        RDATA_S2,
        WDATA_M1,
        WDATA_M2,
        resp,
        RDATA,
        HADDR,
        sel_0,
        sel_1,
        sel_slave,
        sel_2,
        WDATA
        );

Endmodule
```

2.5. Top Module Source Code

```
module top_module(
    input wire [1:0] resp0,
    input wire [1:0] resp1,
    input wire [1:0] resp2,
    input wire [13:0] HADDR_M1,
    input wire [13:0] HADDR_M2,
    input wire [31:0] RDATA_S0,
    input wire [31:0] RDATA_S1,
    input wire [1:0] sel,
    input wire [31:0] RDATA_S2,
    input wire [31:0] WDATA_M1,
    input wire [31:0] WDATA_M2,
    output wire [1:0] resp,
    output wire [31:0] RDATA,
    output wire [13:0] HADDR,
    output wire sel_0,
    output wire sel_1,
    output wire [1:0] sel_slave,
    output wire sel_2,
    output wire [31:0] WDATA

    );
    mux2_1 ADD_MUX(HADDR_M1,HADDR_M2,sel,HADDR);
    Decoder1_3 Dec(HADDR,sel_0,sel_1,sel_2,sel_slave);
    Mux_3_1 ReadMux(RDATA_S0,RDATA_S1,RDATA_S2,sel_slave,RDATA);
    Mux_3_1_Resp RespMux(resp0,resp1,resp2,sel_slave,resp);
    Mux_2_1_Write_Data WriteDataMux(WDATA_M1,WDATA_M2,sel,WDATA);
endmodule
```

2.6. Top Module & Bus Module Diagram

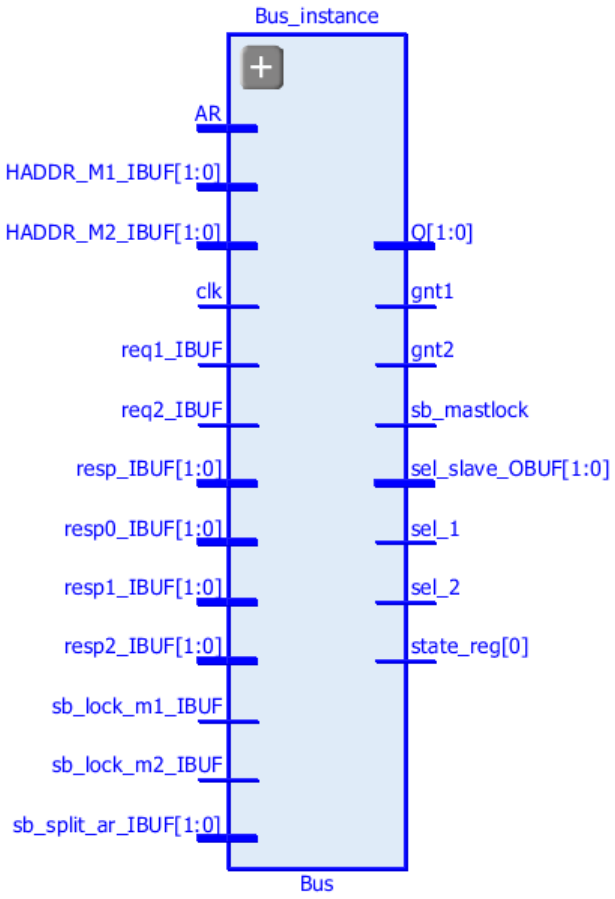


Figure 2-3 – Bus module diagram

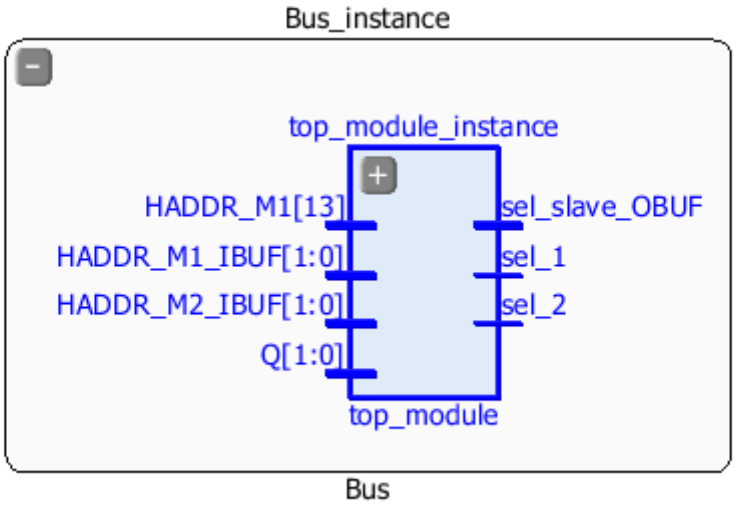


Figure 2-4 – Top module diagram

3. Address Decoder Verification

3.1. Address Decoder Source Code

```
module Decoder1_3(
inp_Addr,// This is the input address from the Address mux
rst,//Reset
sel_s0,// Select pin that goes to slave0 this goes high when slave 0 selected
sel_s1,//Select pin that goes to slave1 this goes high when slave 1 selected
sel_s2,//Select pin that goes to slave2 this goes high when slave 2 selected
sel_slave//This is a indication to ReadMux and RespMux about the slave
selected 00-Slave 0    01-Slave 1    10 -Slave 2
    );

 input [13:0] inp_Addr;
 input rst;
 output sel_s0;
 output sel_s1;
 output sel_s2;
 output [1:0] sel_slave;

 reg sel_s0;
 reg sel_s1;
 reg sel_s2;
 reg [1:0] sel_slave;



 always @ (inp_Addr or rst)
 begin : MUX
   if (inp_Addr[13:12] == 2'b00 && rst != 1) begin
       sel_s0=1;
       sel_s1=0;
       sel_s2=0;
       sel_slave=0;
   end else if(inp_Addr[13:12] == 2'b01  && rst != 1) begin
       sel_s0=0;
          sel_s1=1;
          sel_s2=0;
       sel_slave=1;
   end else if(inp_Addr [13:12]== 2'b10 && rst != 1)
        begin
        sel_s0=0;
                  sel_s1=0;
                  sel_s2=1;
        sel_slave=2;
        end
        else if(rst ==1)
        begin
        sel_s0=0;
                sel_s1=0;
                  sel_s2=0;
        sel_slave=3;

        end
        else
        begin
        sel_s0=1;
               sel_s1=0;
               sel_s2=0;
```

```
                sel_slave=0;
        end


 end
endmodule
```

3.2. Address Decoder Test Cases

1. Address is to Slave 0 (inp_Addr=0'b00111111111111)

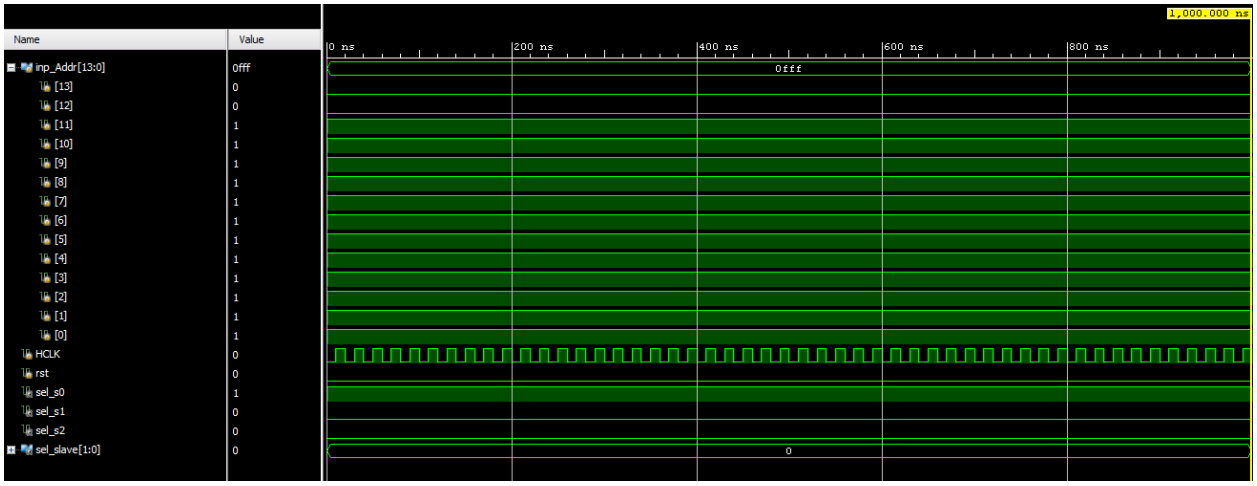


Figure 3-1 – Test case 1

2. Address is to Slave 1 (inp_Addr=0'b01111111111111)

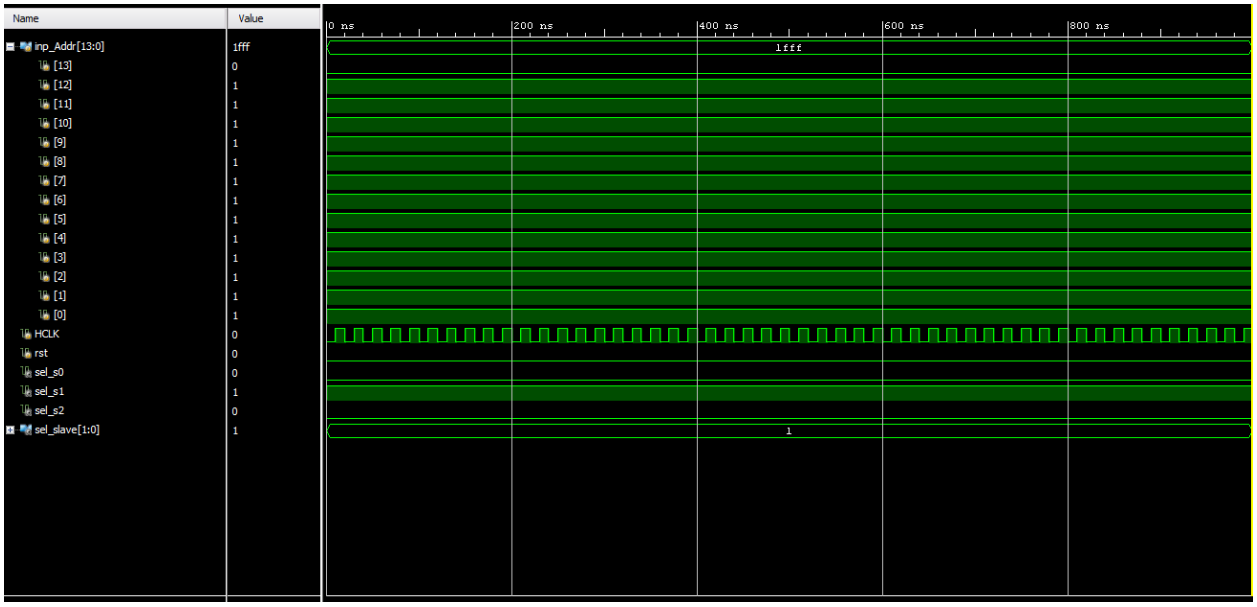


Figure 3-2 – Test case 2

3. Address is to Slave 2 (inp_Addr=0'b10111111111111)

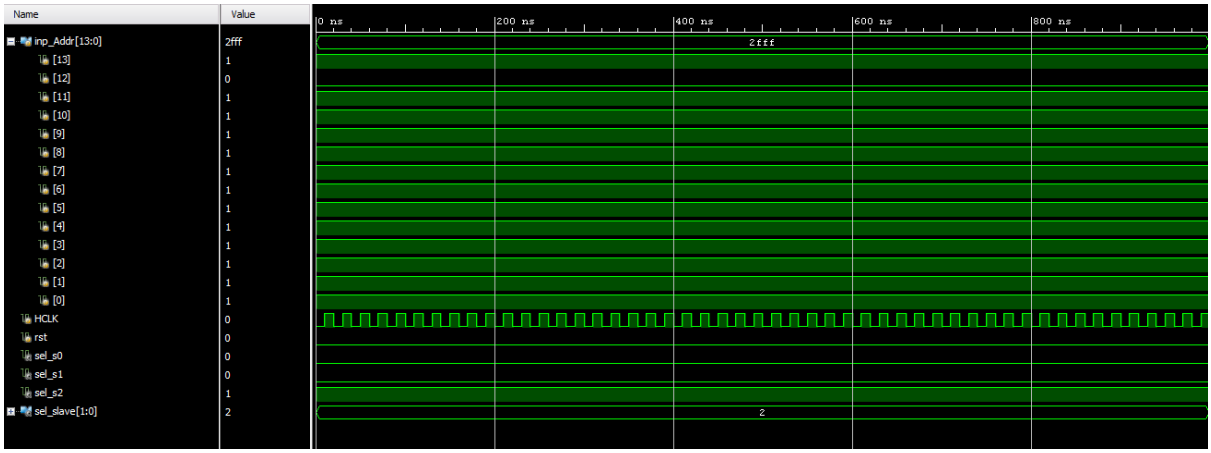


Figure 3-3 – Test case 3

4. Reset Verification (rst = 1)

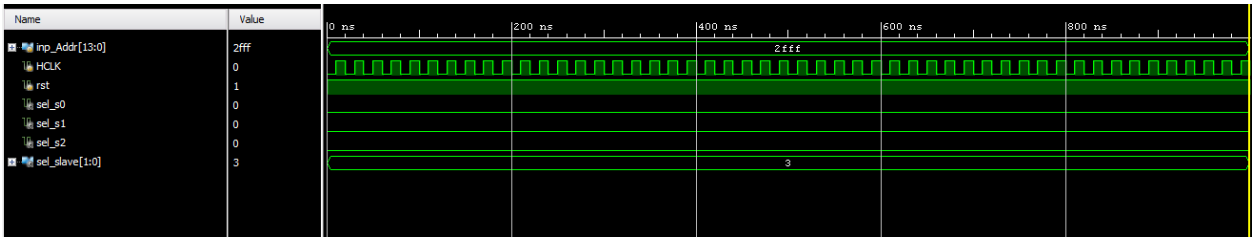


Figure 3-4 – Test case 4

3.3. Address Decoder Diagram

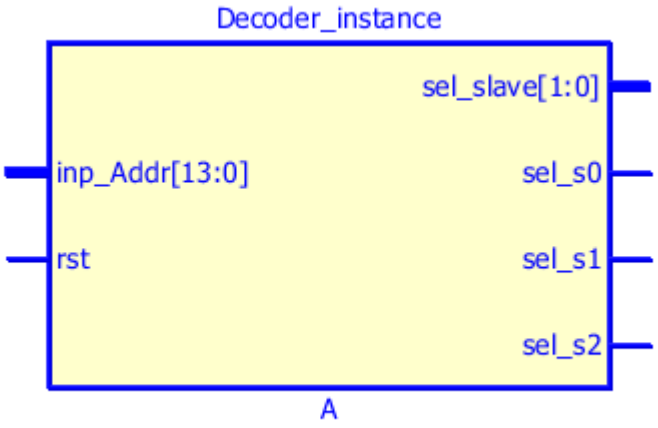


Figure 3-5 - Address decoder diagram

4. Bus Master Verification

4.1. Master Source Code

```
//Transfer types
`define NON_SEQ     2'd0
`define SEQ         2'd1
`define BUSY        2'd2
`define IDLE_TRANS  2'd3

//Response types
`define OKAY  2'b00
`define ERROR 2'b01
`define RETRY 2'b10
`define SPLIT 2'b11

//Burst types
`define BURST_SINGLE 2'b00
`define BURST_INCR4 2'b01
`define BURST_INCR8 2'b10

//width of the transfer
`define SIZE_BYTE 2'b00
`define SIZE_HALFWORD 2'b01
`define SIZE_WORD 2'b10

module Master (
    input clk,
    input resetn,

    input b_grant,
    input b_ready,
    input [1:0] b_resp,
    input [31:0] b_rData,

    output reg b_req,
    output reg b_lock,
    output reg [1:0] b_trans,
    output reg [4:0] b_control,
    output reg [13:0] b_addr,
    output reg [31:0] b_wData,

    //inputs for testing
    input u_req,
    input u_lock,
    input [1:0] u_transSize,
    input [1:0] u_burst,
    input u_write,
    input [13:0] u_addr,
    input [31:0] u_wData

    );

    //local params
    localparam STATE_WIDTH = 3;
    localparam [STATE_WIDTH-1:0] STATE_IDLE        = 0;
```

```
localparam [STATE_WIDTH-1:0] STATE_REQ         = 1;
localparam [STATE_WIDTH-1:0] STATE_TRANS_BEGIN = 2;
localparam [STATE_WIDTH-1:0] STATE_TRANS       = 3;
localparam [STATE_WIDTH-1:0] STATE_TRANS_END   = 4;
localparam [STATE_WIDTH-1:0] STATE_SPLIT       = 5;


reg [3:0] beat_counter;
reg [13:0] Addr;
reg [31:0] RData; //to read the data from b_rData
reg [2:0]  Addr_inc;

reg [STATE_WIDTH-1:0] state;

initial begin
    state = STATE_IDLE;  //initially idle state
end

always @(posedge clk) begin
    if(!resetn) begin
        beat_counter <= 4'b0;
        b_req        <= 1'b0;
        b_lock       <= 1'b0;  //master has lock
        b_trans      <= `IDLE_TRANS;  // in idle state
        b_addr       <= 14'b0;
    end
    else begin
        if(u_req && state==STATE_IDLE) state = STATE_REQ;

        case(state)
            STATE_IDLE: begin
                beat_counter <= 4'b0;
                b_req        <= 1'b0;
                b_lock       <= 1'b0;  //master has lock
                b_trans      <= `IDLE_TRANS; // in idle state
                b_addr       <= 14'b0;
            end

            STATE_REQ : begin
                b_req <= u_req;
                b_lock <= u_lock;
                if(b_grant) begin
                    state <= STATE_TRANS_BEGIN;
                    Addr  <= u_addr;
                    b_req <= 0;
                    b_lock <= 0;   //master has no lock

                    case(u_burst)  //burst mode
                        `BURST_SINGLE: beat_counter <= 4'b0001;
                        `BURST_INCR4  : beat_counter <= 4'b0100;
                        `BURST_INCR8  : beat_counter <= 4'b1000;
                    endcase

                    case(u_transSize) // HSIZE
                        `SIZE_BYTE    : Addr_inc <= 32'd1;
                        `SIZE_HALFWORD: Addr_inc <= 32'd2;
                        `SIZE_WORD    : Addr_inc <= 32'd4;
```

```
                    endcase
                end
            end

            STATE_TRANS_BEGIN: begin
                if(b_ready) begin
                    b_trans <= `NON_SEQ; // non sequence state in trans
begin
                    b_addr  <= Addr;
                    b_control <= {u_write,u_transSize,u_burst}; //
write-1bit transsize-2bit burst-2bit

                    Addr <= Addr + Addr_inc;
                    beat_counter <= beat_counter-1;

                    state <= (beat_counter-1 == 4'b0)?
STATE_TRANS_END:STATE_TRANS;
                end

            end

            STATE_TRANS: begin
                if(b_ready && b_resp == `OKAY) begin
                   b_trans <= `SEQ;  // sequence  in trans
                   b_addr  <= Addr;
                   Addr <= Addr + Addr_inc;
                   b_control <= {u_write,u_transSize,u_burst};
                   beat_counter <= beat_counter -1;

                   state <= (beat_counter-1 == 4'b0)?
STATE_TRANS_END:STATE_TRANS;

                   if(u_write) b_wData <= u_wData; //write the data
                   else RData <= b_rData; //read the data
                end
                else if(b_resp == `ERROR) state <= STATE_IDLE;
                else if(b_resp == `SPLIT) state <= STATE_SPLIT;
            end

            STATE_TRANS_END: begin
               if(b_ready && b_resp == `OKAY) begin
                   beat_counter <= 4'b0;
                   b_req        <= 1'b0;
                   b_lock       <= 1'b0; //master has lock
                   b_trans      <= `IDLE_TRANS;  //idle in end of trans
                   b_addr       <= 14'b0;
                   state        <= STATE_IDLE;
                   if(u_write) b_wData <= u_wData; //write the data
                   else RData <= b_rData; //read the data
               end
               else if(b_resp == `ERROR) state <= STATE_IDLE;
               else if(b_resp == `SPLIT) state <= STATE_SPLIT;
            end

            STATE_SPLIT: begin
                if(b_grant && beat_counter==4'b0) state <=
STATE_TRANS_END;
```

```
                    else if(b_grant) state <= STATE_TRANS;
                end
            endcase
        end

    end

endmodule
```

4.2. Master Test Cases

1. Write operation

```
module writeOperation();

    reg clk;
    reg resetn;

    reg b_grant;
    reg b_ready;
    reg [1:0] b_resp;
    reg [31:0] b_rData;

    wire b_req;
    wire b_lock;
    wire [1:0] b_trans;
    wire [4:0] b_control;
    wire [13:0] b_addr;
    wire [31:0] b_wData;

    reg u_req;
    reg u_lock;
    reg [1:0] u_transSize;
    reg [1:0] u_burst;
    reg u_write;
    reg [13:0] u_addr;
    reg [31:0] u_wData;

    Master master(.clk(clk), .resetn(resetn), .b_grant(b_grant),
.b_ready(b_ready), .b_resp(b_resp), .b_rData(b_rData),
                    .b_req(b_req), .b_lock(b_lock), .b_trans(b_trans),
.b_control(b_control),
                    .b_addr(b_addr), .b_wData(b_wData), .u_req(u_req),
.u_lock(u_lock), .u_transSize(u_transSize),
                    .u_burst(u_burst), .u_write(u_write), .u_addr(u_addr),
.u_wData(u_wData) );

    always #1 clk=~clk;

    initial begin
        clk<=0;
        resetn<=0;
        #5 resetn<=1;
    //-------------------------------------------------------------------
    //write operation
```

```
        u_req = 1;
        u_lock = 0;
        u_transSize = 2'b10;
        u_burst = 2'b01;
        u_write = 1; // write operation
        u_addr = 32'd10;
        u_wData = 32'd15;
        b_grant = 1;
        b_resp = 2'b00;
        b_ready = 1;
    end

endmodule
```

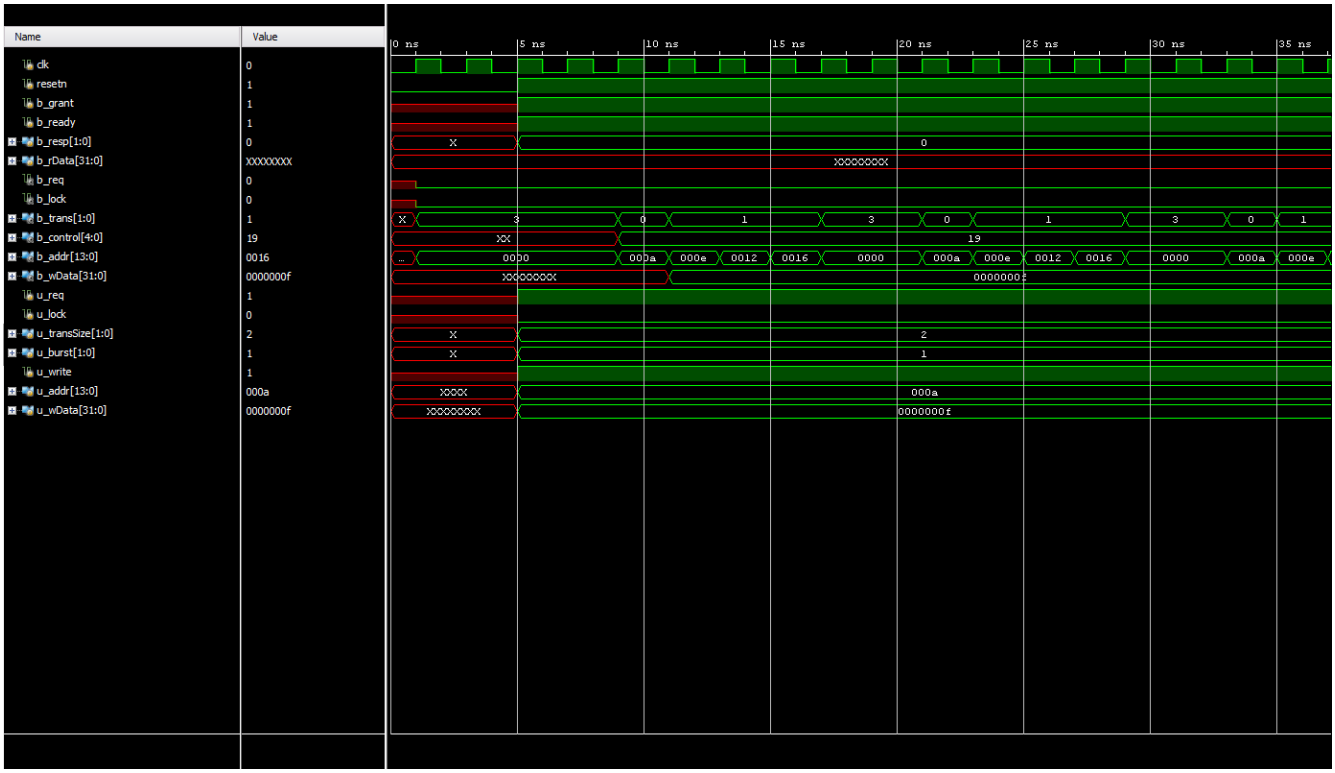


Figure 4-1 – Write operation simulation diagram

Input: 32'd15

Output: b_wData (0000000f)

1. Read operation

```
module readOperation();

reg clk;
reg resetn;

reg b_grant;
reg b_ready;
reg [1:0] b_resp;
reg [31:0] b_rData;

wire b_req;
wire b_lock;
wire [1:0] b_trans;
wire [4:0] b_control;
wire [13:0] b_addr;
wire [31:0] b_wData;

reg u_req;
reg u_lock;
reg [1:0] u_transSize;
reg [1:0] u_burst;
reg u_write;
reg [13:0] u_addr;
reg [31:0] u_wData;

Master master(.clk(clk), .resetn(resetn), .b_grant(b_grant),
.b_ready(b_ready), .b_resp(b_resp), .b_rData(b_rData),
                .b_req(b_req), .b_lock(b_lock), .b_trans(b_trans),
.b_control(b_control),
                .b_addr(b_addr), .b_wData(b_wData), .u_req(u_req),
.u_lock(u_lock), .u_transSize(u_transSize),
                .u_burst(u_burst), .u_write(u_write), .u_addr(u_addr),
.u_wData(u_wData) );

always #1 clk=~clk;

initial begin
    clk<=0;
    resetn<=0;
    #5 resetn<=1;
//-------------------------------------------------------------------------
//read operation
    u_req = 1;
    u_lock = 0;
    u_transSize = 2'b10;
    u_burst = 2'b01;
    u_write = 0; // read operation
    u_addr = 32'd10;
    b_grant = 1;
    b_resp = 2'b00;
    b_ready = 1;
    b_rData = 32'd16; //read Data Input
end

endmodule
```

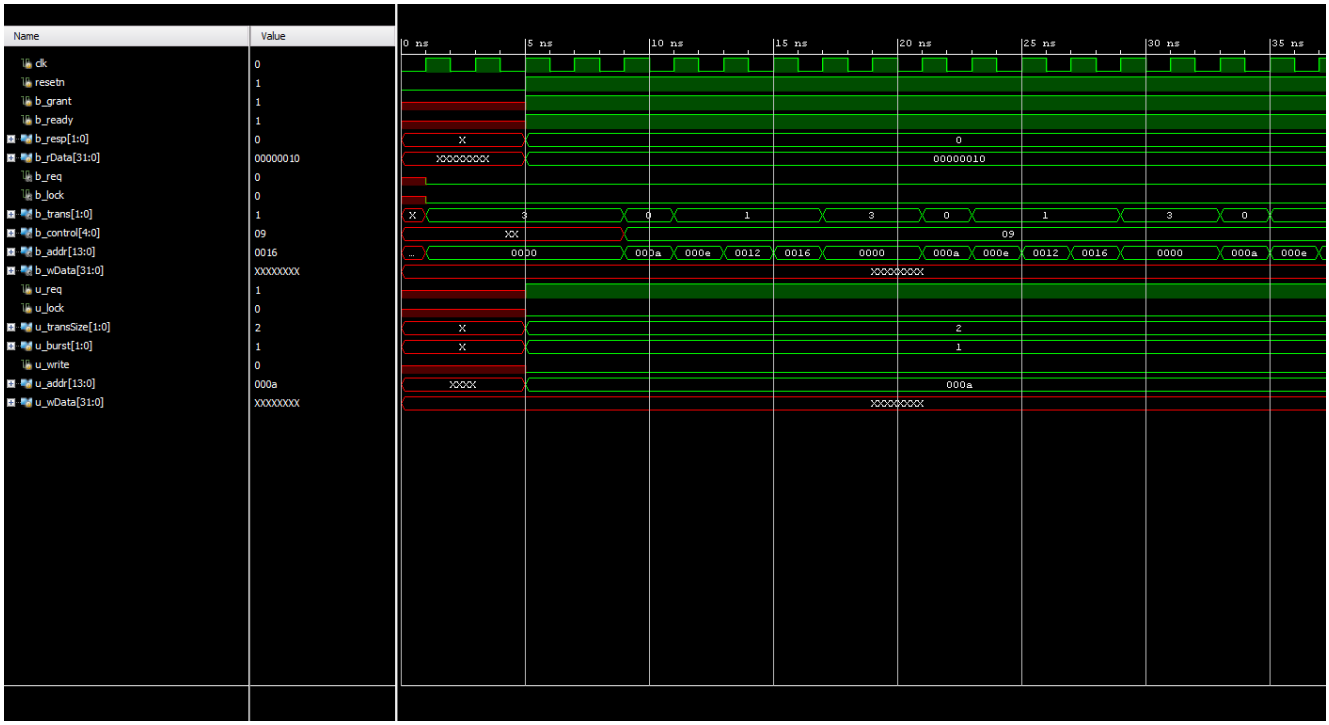


Figure 4-0-1 – Read operation simulation diagram

Input: 32'd10

Output: u_wData(00000010)

5. Slave Verification

There are three slaves for this bus design. Two of them has 2 K memory and the other one has 4 K memory. To recognize these slaves separately we use extra two bits from the master. Using a decoder, we recognize the relevant slave. Then the slave receives the relevant address. There is an algorithm in the Slave to remove first two bits and get the address. This slave does not support for the burst operation. This only support for the single transfer. This supports for the split transaction. We modelled that using a memory address. When master asks for the data in that memory address slave gives a split response.

5.1. Split Response Timing diagram

Here the master asks for the address 0'b00000000000. When Master asks for this address Slave gives a Split response.

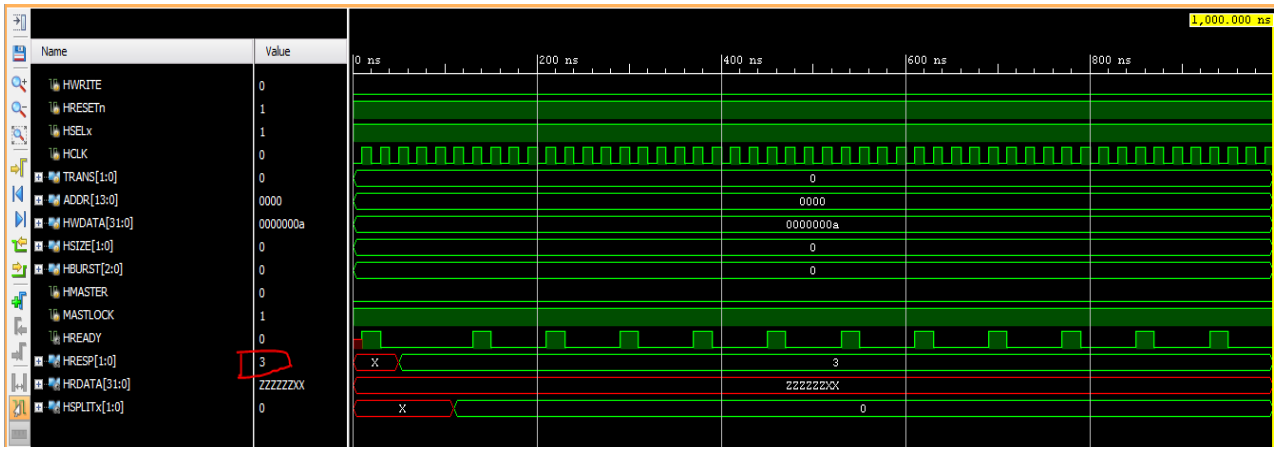


Figure 5-1 – Split response simulation

5.2. Write Function Timing Diagram

Input Address: reg [13:0] ADDR=0'b00000000000001;

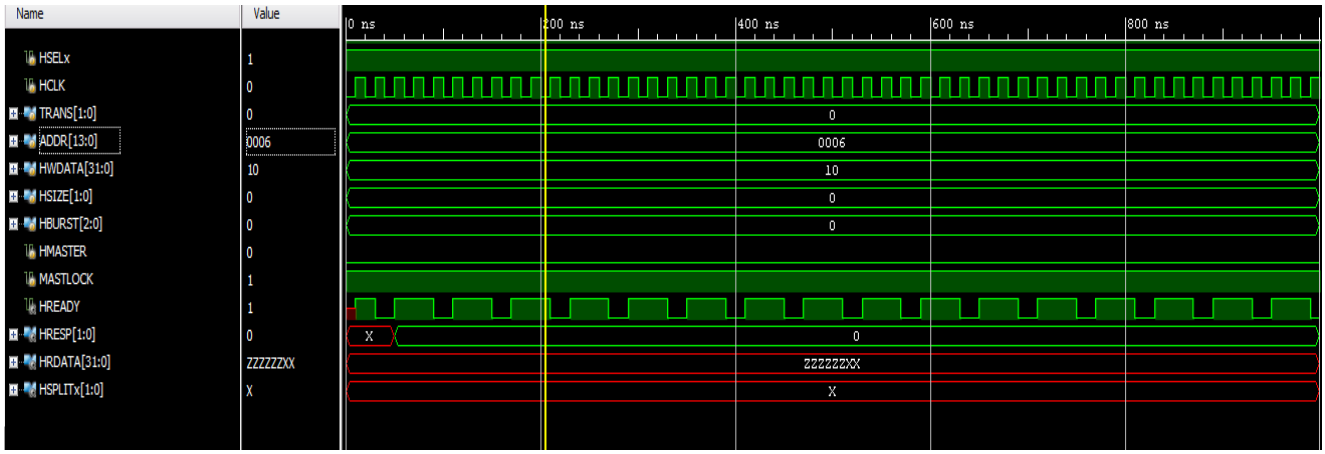


Figure 5-2 – Write function simulation

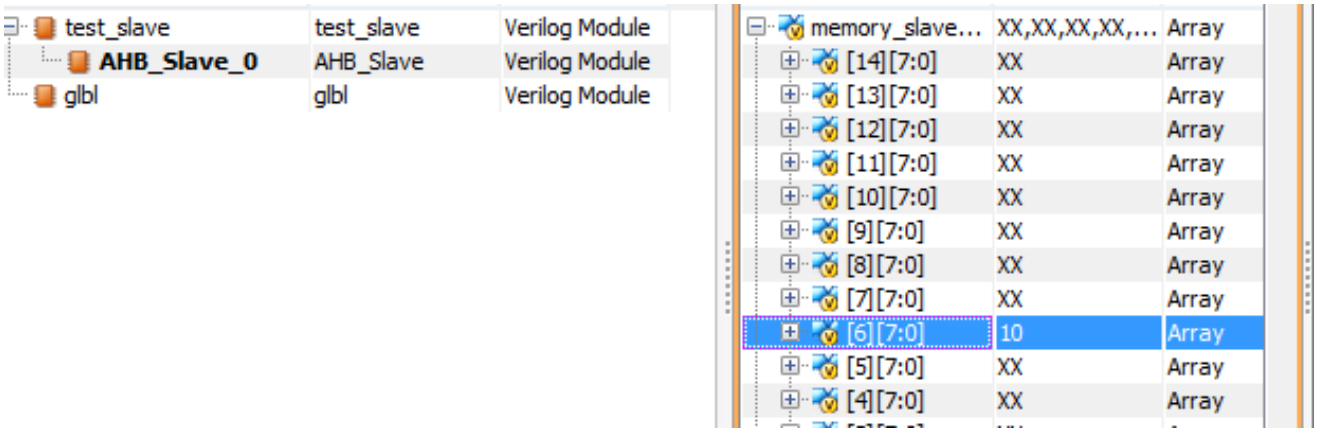


Figure 5-3 Memory locations in Slave

5.3. Read Function Timing Diagram

Input Address: reg [13:0] ADDR=0'b00000000000001;

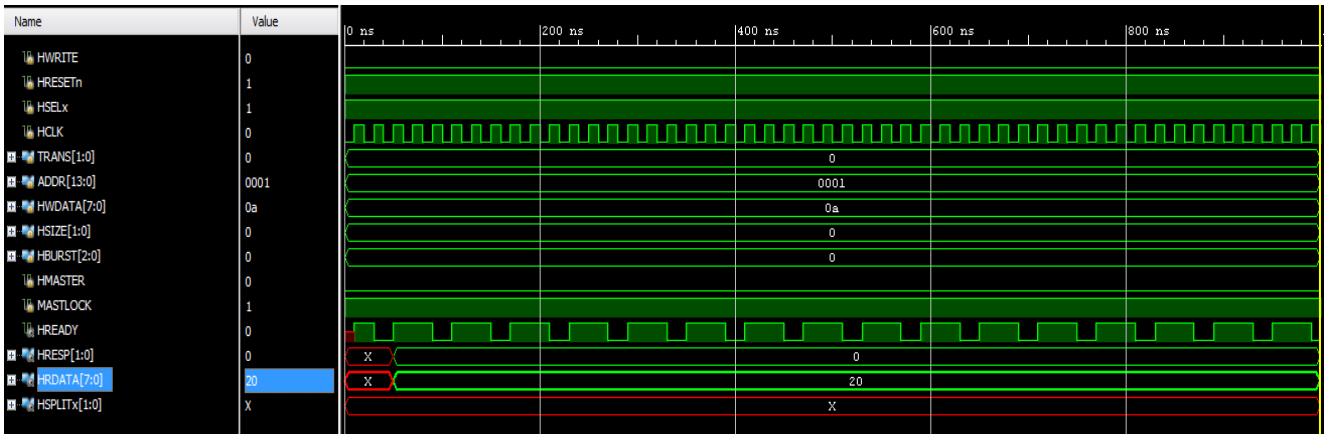


Figure 5-4 – Read function simulation

5.4. Slave Source Code

```
`define IDLE         3'b000
`define ACTIVE       3'b001
`define AGAIN        3'b010
`define LITTLE       3'b011
`define TIME_PASS    3'b111
`define WRITE_BURST 3'b100
`define READ_BURST   3'b101


`define NON_SEQ      2'd0
`define SEQ         2'd1
`define BUSY        2'd2
`define IDLE_TRANS   2'd3

`define OKAY   2'b00
`define ERROR 2'b01
`define RETRY 2'b10
`define SPLIT 2'b11



module AHB_Slave( HREADY,//output signal to the arbiter
                  HRESP,//responce of the Slave to the arbiter 11-Split
                  HRDATA,//Read data from slave to Read Data Mux
                  HSPLITx,//Splitx  signal  that  request  the  master  to
arbiter
                           HSELx,//Selection  input  that  is  given  by
decoder to the Slave
                  ADDR,//Address
```

```
                    HWRITE,//Write signal if this is zero that means its read
operation
                    TRANS,//This is not used in our simulation
                    HSIZE,//This is not used in our simulation
                    HBURST,//This is not used in our simulation
                    HWDATA,//Write data from write data mux
                    HRESETn,//Reset signal
                    HCLK,//Clock
                    HMASTER,//Master if 0- Master1 else master2
                    MASTLOCK//Indication whether their is a lock in current
transaction
);

output  HREADY;
output [1:0] HRESP;
output [7:0] HRDATA;
output [1:0] HSPLITx;
input MASTLOCK;//Master lock 0-Master has no Lock 1- Master Has lock
input [1:0] TRANS;//Input value of Trans 00-Non Seq 01-Seq 11-Idle 10-Busy
input HSELx,HWRITE,HRESETn,HCLK;
input [13:0] ADDR;//14 Bit address First two bits to identify slave next 12
bits for addr in 11 bit addr slave first of 12 bit becomes zero
reg HMASTLOCK;
reg [11:0] HADDR;
input [7:0] HWDATA;
input  [1:0] HSIZE;//00- 8 bit 01- 16 bit 10 - 32 bit
input [2:0] HBURST;
input  HMASTER;//master 1 -      master 2 -1
reg [1:0] HTRANS;
reg [7:0] HRDATA;

reg  HREADY;
reg  [1:0] HRESP;
reg  [1:0] HSPLITx;
reg  [4:0] local_addr;
reg  [3:0]SPLIT_RESP;
reg [7:0] memory_slave [2047:0]; //This is a 2K memory slave
reg[2:0] ps_slave1,ns_slave1;
initial
begin
ns_slave1=`IDLE;
memory_slave[8'b00000000]=8'd10;//Hardcoed Memory Values
memory_slave[8'b00000001]=8'd20;
memory_slave[8'b00000010]=8'd30;
HMASTLOCK=MASTLOCK;
end
integer count;


always @ (ADDR or MASTLOCK or ps_slave1 or ns_slave1 or HRESETn or
           HSELx or HWDATA or HWRITE )

begin
HADDR=ADDR[10:0];

case (ps_slave1)

`IDLE  :begin

         if(!HRESETn && HSELx==0)
         ns_slave1=`IDLE;
```

```
          else
          begin
         // HSELx=1'd1;
         HREADY=1'b1;
    //HMASTLOCK=1'b0;
       //  HWRITE=1'b1;
         //HBURST=3'b001;
         // HADDR=$random %32;
          local_addr=5'b0;
          ns_slave1=`ACTIVE;
end
          end

`ACTIVE : begin
          if(HRESETn && HSELx && HWRITE && HREADY)
begin
          HREADY=1'b0;
          ns_slave1=`WRITE_BURST;
end
          else if(HRESETn && HSELx && !HWRITE && HREADY)
begin
          HREADY= 1'b0;
          ns_slave1=`READ_BURST;
end
          else if(!HREADY)
begin
          ns_slave1=`AGAIN;
        HRESP= `RETRY;
end
          else
          ns_slave1=`IDLE;
 end

`AGAIN :  begin
          if(HREADY)
          ns_slave1=`ACTIVE;
          else
          ns_slave1=`LITTLE;
          end

`WRITE_BURST  : begin
          if (HRESETn && HSELx && HWRITE )

          case(HBURST)//Single Transfer is only used
          3'b000 : begin
                                        memory_slave[HADDR]= HWDATA;
                                        HREADY=1'b1; HRESP= `OKAY;
                                        HTRANS=`NON_SEQ;
                                        ns_slave1=`IDLE;
                                    end     //000--Single transfer

                          3'b001  : begin   // incrememting Burst unspecified
Length
                                         memory_slave[HADDR]=HWDATA;

                                         HADDR=HADDR+1;
                                         count=count+1;
                                            if(count<32)
                                                ns_slave1=`WRITE_BURST;
                                            else
```

```
                                                HREADY=1'b1;          HRESP=
`OKAY;
                                ns_slave1=`IDLE;

                                        end//001

                              3'b010 : begin    // 4BEAT WRAPPING burst

memory_slave[HADDR]=HWDATA;
                                                HADDR=HADDR+4;
                                               count=count+1;
                                                if(count==4)
                                                begin
                                  HREADY=1'b1; HRESP= `OKAY;
                                                 HADDR=local_addr;
                                                    count=0;
                                                 ns_slave1=`IDLE;
                                                end//count<4
                                             else
                                 ns_slave1=`WRITE_BURST;
                                     end//010

                                3'b011 : begin  ///4 beat Incrementing Burst
                                                memory_slave[HADDR]=HWDATA;
                                                HADDR=HADDR+4;
                                                count=count+1;
                                                if(count<4)
                                                    ns_slave1=`WRITE_BURST;
                                                else
                                HREADY=1'b1;
                                                  HRESP= `OKAY;
                                                    ns_slave1=`IDLE;
                                                 end//011

                                3'b100 : begin          // 8 Beat Wrapping Burst
                                                   memory_slave[HADDR]=HWDATA;
                                                   HADDR=HADDR+4;
                                                  count=count+1;
                                                   if(count==8)
                                                   begin
                                 HREADY=1'b1; HRESP= `OKAY;
                                                    HADDR=local_addr;
                                                       count=0;
                                                    ns_slave1=`IDLE;
                                                   end//count<4
                                                else
                                 ns_slave1=`WRITE_BURST;
                                     end//100


                                3'b101 : begin  ///8 beat Incrementing Burst
                                                memory_slave[HADDR]=HWDATA;
                                                HADDR=HADDR+4;
                                                count=count+1;
                                                if(count<8)
                                                    ns_slave1=`WRITE_BURST;
                                                else
                                                    HREADY=1'b1;HRESP= `OKAY;
                                                    ns_slave1=`IDLE;

                                            end//101
```

```
                        3'b110 : begin // 16 beat wrapping Burst
                                    memory_slave[HADDR]=HWDATA;
                                    HADDR=HADDR+4;
                                   count=count+1;
                                    if(count==16)
                                    begin
                          HREADY=1'b1;HRESP= `OKAY;
                                     HADDR=local_addr;count=0;
                                     ns_slave1=`IDLE;
                                    end//count<4
                                 else
                         ns_slave1=`WRITE_BURST;

                                 end//110

                        3'b111 : begin
                                     memory_slave[HADDR]=HWDATA;
                                     HADDR=HADDR+4;
                                     count=count+1;
                                     if(count<16)
                                         ns_slave1=`WRITE_BURST;
                                     else
                                         HREADY=1'b1;HRESP= `OKAY;
                                         ns_slave1=`IDLE;
                               end//111

                             default : begin
                            HREADY=1'b1;HRESP= `OKAY;
                            ns_slave1=`IDLE;
                           end
                          endcase//for WRITE operation
          else
           HRESP= `ERROR;
                end//if(WRIte operation)

`READ_BURST :
                       //READ Operation Starts Here
                       begin
              if(HRESETn && HSELx && !HWRITE)
                        case(HBURST)

                    3'b000 : begin//This is the only used burst operation
                                    if(HADDR!=0)//This   is   made   to
demontrate the split operation is the data in 0 th address is asked then as
the response slave gives a Split responce
                                    begin
                                    HRDATA=memory_slave[HADDR];
                                    HREADY=1'b1;
                                     ns_slave1=`IDLE;HRESP= `OKAY;
                                    end
                                    else
                                    begin
                                    HREADY=1'b0;
                                     ns_slave1=`LITTLE;HRESP= `SPLIT;
                                    end

                                 end    //000--Single transfer

                     3'b001  : begin   // incrememting Burst unspecified
Length
```

```
                                        HRDATA=memory_slave[HADDR];
                                        HADDR=HADDR+1;
                                        count=count+1;
                                           if(count<32)
                                               ns_slave1=`READ_BURST;
                                           else
                                                HREADY=1'b1;HRESP= `OKAY;
                                                ns_slave1=`IDLE;

                                    end//001

                                3'b010 : begin    // 4BEAT WRAPPING burst

HRDATA=memory_slave[HADDR];
                                                 HADDR=HADDR+4;
                                               count=count+1;
                                                if(count==4)
                                                begin
                                                  HREADY=1'b1;
                                  HADDR=local_addr;count=0;
                                                 ns_slave1=`IDLE;
                                                end//count<4
                                           else
                                 ns_slave1=`READ_BURST;
                                     end//010

                                3'b011 : begin  ///4 beat Incrementing Burst
                                                HRDATA=memory_slave[HADDR];
                                                HADDR=HADDR+4;
                                                count=count+1;
                                                if(count<4)
                                                    ns_slave1=`READ_BURST;
                                                else
                                                     HREADY=1'b1;HRESP= `OKAY;
                                 ns_slave1=`IDLE;
                                                 end//011

                                3'b100 : begin          // 8 Beat Wrapping Burst
                                                    HRDATA=memory_slave[HADDR];
                                                    HADDR=HADDR+4;
                                                  count=count+1;
                                                   if(count==8)
                                                   begin
                                                     HREADY=1'b1;HRESP= `OKAY;
                                     HADDR=local_addr;
                                                              count=0;
                                                    ns_slave1=`IDLE;
                                                   end//count<4
                                           else
                                 ns_slave1=`READ_BURST;
                                     end//100


                                3'b101 : begin  ///8 beat Incrementing Burst
                                                HRDATA=memory_slave[HADDR];
                                                HADDR=HADDR+4;
                                                count=count+1;
                                                if(count<8)
                                                    ns_slave1=`READ_BURST;
                                                else
                                                     HREADY=1'b1;HRESP= `OKAY;
```

```
                                ns_slave1=`IDLE;

                                        end//101

                              3'b110 : begin // 16 beat wrapping Burst
                                              HRDATA=memory_slave[HADDR];
                                              HADDR=HADDR+4;
                                             count=count+1;
                                              if(count==16)
                                              begin
                                                HREADY=1'b1;HRESP= `OKAY;
                                HADDR=local_addr;
                                                      count=0;
                                               ns_slave1=`IDLE;
                                              end//count<4
                                           else
                                ns_slave1=`READ_BURST;

                                        end//110

                              3'b111 : begin
                                           HRDATA=memory_slave[HADDR];
                                           HADDR=HADDR+4;
                                           count=count+1;
                                           if(count<16)
                                               ns_slave1=`READ_BURST;
                                           else
                                                HREADY=1'b1;
                                ns_slave1=`IDLE;
                                       end//111

                              default :  begin
                              HREADY=1'b1;HRESP= `OKAY;
                              ns_slave1=`IDLE;
                              end

                           endcase //for Read Operation
                  else
                    HRESP= `ERROR;
                  end
         `LITTLE : begin
                  SPLIT_RESP=HMASTER;
                  if(HMASTLOCK)
                  ns_slave1=`TIME_PASS;
                  else
                  begin
                  HSPLITx=SPLIT_RESP;
                  ns_slave1=`IDLE;
                  end
                  end
         `TIME_PASS :begin//To cover additional clock cycle before splitx
                 ns_slave1=`LITTLE;
                 HMASTLOCK=0;
         end
         endcase
         end
         always@(posedge HCLK)
         begin
         ps_slave1=ns_slave1;
         end
         endmodule
```

6. Arbiter Verification

6.1. Arbiter Source Code

```
module arbiter(

clk,   // clock input
rst,   // reset input
req1,  // request signal of master 1
sb_lock_m1, //lock signal of master 1
req2,// request signal of master 2
sb_lock_m2,//lock signal of master 2
sb_split_ar, //SplitX signal from salves,LSB-reresent master 1,MSB reresent
Master 2
sb_resp_ar, // resonse signal from slaves,00-Okay,01-error,11-slit,10-retry
gnt1, // grant signal to master 1
gnt2, //grant signal to master 2
sb_masters, //2 bit signal to slaves,01- master 1,10- master 2
sb_mastlock  // one bit signa
 );
// parameter definitions
endmodule
//-------------------------------------------------------------------------
-----------------------------------------
// localparam definitions
    localparam                          SB_ADDR_WIDTH         = 32;
    localparam                          SB_TRAS_TYPE          = 2;
    localparam                          SB_BURST_NUM          = 3;
    localparam                          SB_RESP_TYPE          = 2;
    localparam                          SB_NUM_MASTER         = 2;
    localparam                          SB_SPLIT_NUM_MSTR     = 2;

// I/O signals
//-------------------------------------------------------------------------
-----------------------------------------
 input                                   clk;
input                                   rst;
input                                   req1;
input                                   sb_lock_m1;
input                                   req2;
input                                   sb_lock_m2;
input       [SB_SPLIT_NUM_MSTR-1:0]     sb_split_ar;
input       [SB_RESP_TYPE-1:0]          sb_resp_ar;
output reg                              gnt1;
output reg                              gnt2;
output reg  [SB_NUM_MASTER-1:0]         sb_masters;
output reg                              sb_mastlock;
//----------------------------
parameter SPLIT=2'b11;
parameter idle=2'b00; // idle state of the bus ,no master connect to the
bus
parameter GNT1=2'b01;  // state when master 1 grant the bus
parameter GNT2=2'b10;  //state when master 2 grant the bus
reg [1:0]  state,next_state;
always @ (posedge clk or posedge rst )
begin
if(rst)
state=idle;
```

```
else
state=next_state;
end


//-----------------------
always @ (sb_split_ar)// 10- gnt2 =1 01 gnt1=1
begin
case (state)
idle:begin    //// current state idle
{gnt2 ,gnt1}=sb_split_ar;
sb_masters=sb_split_ar;
sb_mastlock=(sb_lock_m1 | sb_lock_m2);
end
GNT1:begin   //current state ,master 1 grant the bus
{gnt2 ,gnt1}=sb_split_ar;
sb_masters=sb_split_ar; //no master select
sb_mastlock=(sb_lock_m1 | sb_lock_m2);
 end
GNT2:begin //current state ,master 2 grant the bus
{gnt2 ,gnt1}=sb_split_ar;
  sb_masters=sb_split_ar; //no master select
  sb_mastlock=(sb_lock_m1 | sb_lock_m2);
  end
endcase
end
always @ (sb_resp_ar)
begin
case (state)
idle:begin
gnt2=0;
gnt1=0;
sb_masters=2'b00; //no master select
sb_mastlock=0;
end
GNT1:begin  //current state ,master 1 grant the bus
gnt2=1;
gnt1=0;
sb_masters=2'b10;//master 2
sb_mastlock=sb_lock_m2;
end
GNT2:begin  //current state ,master 2 grant the bus
gnt2=0;
gnt1=1;
sb_masters=2'b01;//master 1
sb_mastlock=sb_lock_m1;
end
endcase
end
always @ (state)
begin
case (state)
idle:begin
gnt2=0;
gnt1=0;
sb_masters=2'b00; //no master select
sb_mastlock=0;
end
GNT1:begin  //current state ,master 1 grant the bus
gnt2=0;
gnt1=1;
```

```
sb_masters=2'b01;//master 1
sb_mastlock=sb_lock_m1;
    end
   GNT2:begin  //current state ,master 2 grant the bus
   gnt2=1;
   gnt1=0;
   sb_masters=2'b10;//master 2
   sb_mastlock=sb_lock_m2;
   end
endcase
end // always @ (state)
//-----------------------
 always @ (state,req2,req1,sb_lock_m1,sb_lock_m2)
  begin
 // next_state=0;
case (state)
idle:begin
if(req1)
 next_state=GNT1;
 else if(req2)
 next_state=GNT2;
 else
 next_state=idle;
 end // case: idle
GNT1:begin  //current state ,master 1 grant the bus
 if(sb_resp_ar==SPLIT) //cheack the split signal
  next_state=GNT2;
  else if(sb_lock_m1)
  next_state=GNT1;
 else if(sb_split_ar==2'b10)
  next_state=GNT2;
else if( req1)
next_state=GNT1;
    else
    next_state=GNT2;//default master
      end
GNT2:begin   //current state ,master 2 grant the bus
 if(sb_resp_ar==SPLIT)  //cheack the split signal
    next_state=GNT1;
    else if(sb_lock_m2)
     next_state=GNT2;
    else if(sb_split_ar==2'b01)
     next_state=GNT1;
     else if( req1)
     next_state=GNT1;
     else
     next_state=GNT2;//default master
      end
endcase // case (state)
end // always @ (state,req2,req1)
endmodule  // arbiter
```

‘

6.2. Arbiter Test Cases

1. Verifying the Reset

Test case 1:

Master 1 request the bus

Master 2 not request the bus

Apply a reset

Output:

Master 1 grant the bus

Arbiter reset and master 2 grant the bus

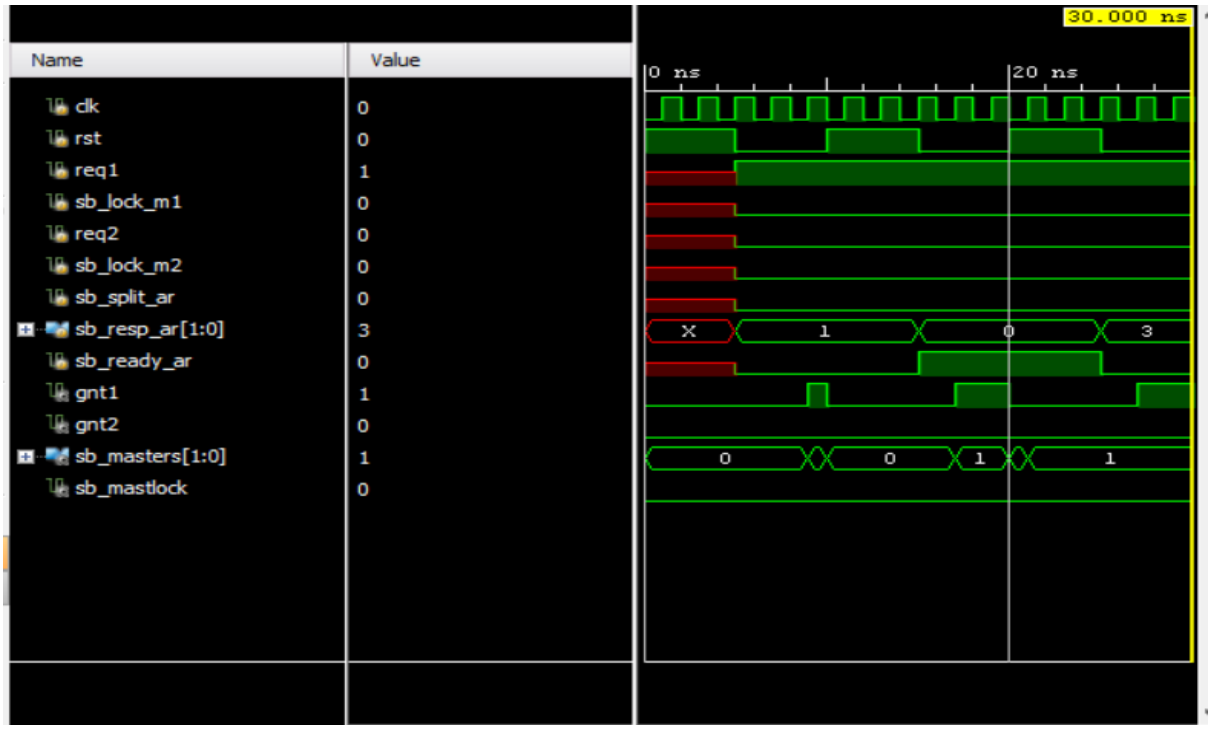


Figure 6 – Reset test for arbiter

2. One master request the bus

Test case 2:

Master 1 request the bus

Master 2 not request the bus

Output:

Master 1 grant the bus

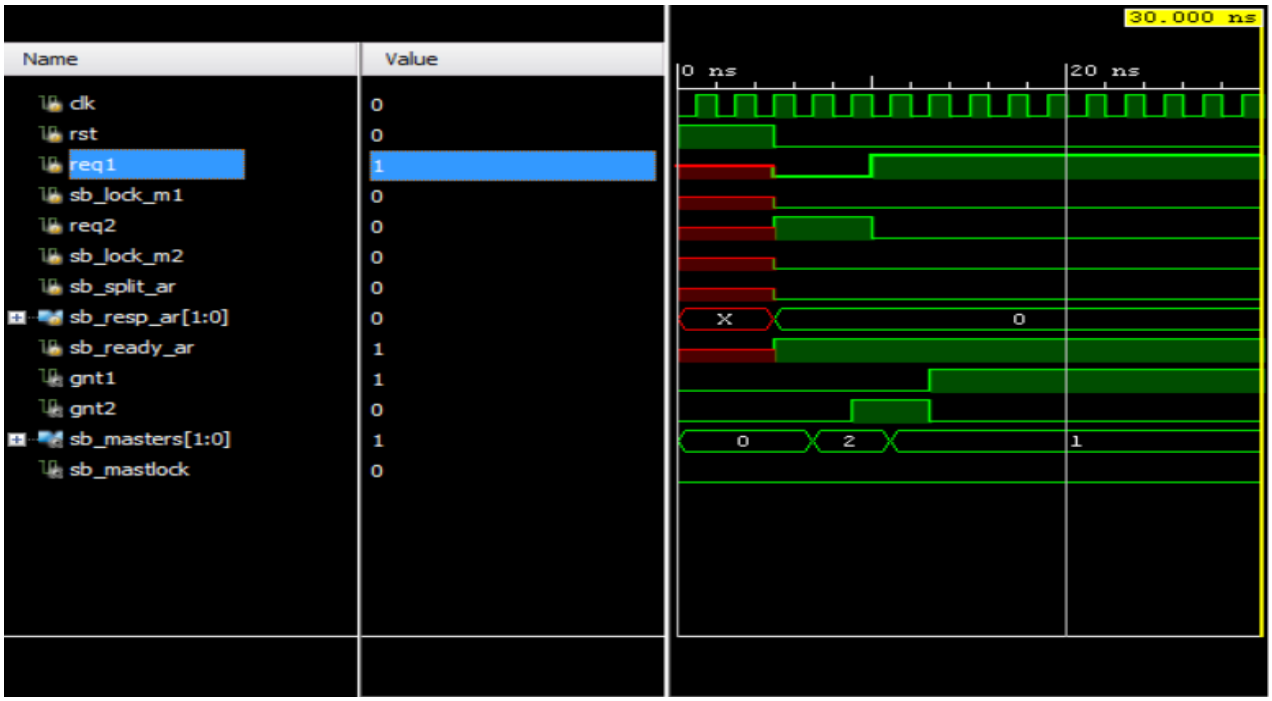


Figure 6.2 – Master request the bus test

3. Both masters request the bus

Test case 3:

Master 1 request the bus

Master 2 request the bus

Output:

Master 1 grant the bus

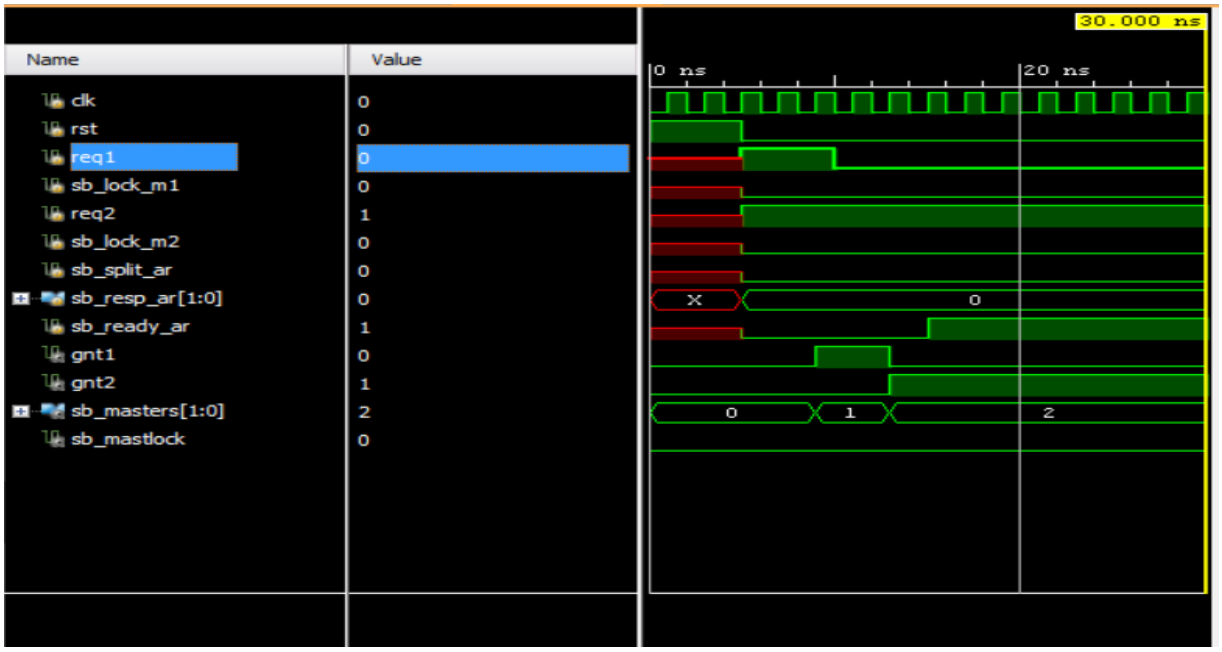


Figure 6.3 – Both master request the bus test

4. Split transaction

Test case 4:

Master 1 request the bus

Master 2 not request the bus

Output:

Master 1 grant the bus

Master grant the bus when split happens

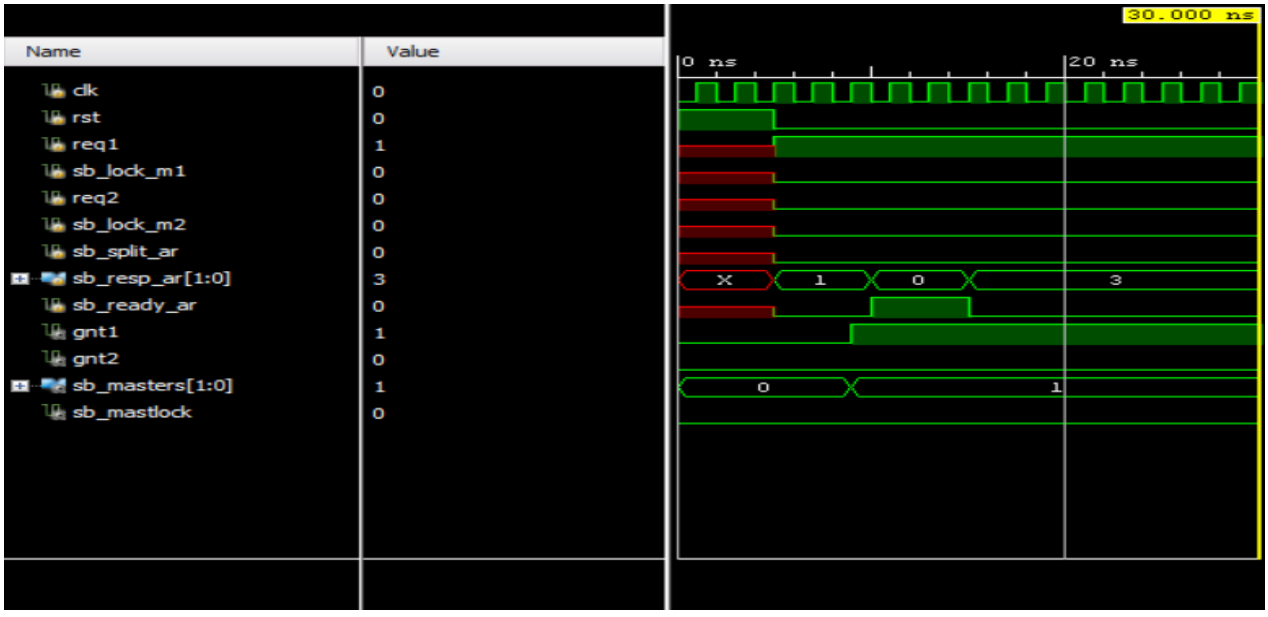


Figure 6.4 – Split transaction test

7. Appendix

7.1. Bus Module Test Bench

```
module Bustest;

reg  clk;
reg  rst;//reset
reg  sb_lock_m1;//lock request from m1 to arbiter
reg  req1;//request to aquire the bus from Master1
reg  req2;//request to aquire the bus from Master2
reg  sb_lock_m2;//lock request from m2 to arbiter
reg  [1:0] resp;//Responce from mux
reg  [1:0] resp0;//responce from slave 0
reg  [1:0] resp1;//responce from slave 1
reg  [1:0] resp2;//responce from slave 2
reg  [13:0] HADDR_M1;//Address of Master 1
reg  [13:0] HADDR_M2;//Address of Master 2
reg  [31:0] RDATA_S0;//Read Data from Slave 0
reg  [31:0] RDATA_S1;//Read Data from Slave 1
reg  [31:0] RDATA_S2;//Read data from Slave 2
reg  [31:0] WDATA_M1;//Write data from M1
reg  [31:0] WDATA_M2;//Write data from M2
reg  [1:0] sb_split_ar;//Splitx from slave to arbiter 01-GNT Master 1 // 10-
GNT Mater 2

wire gnt1;// Grant for Master 1
wire gnt2;//Grnt for Master 2
 wire [1:0] sb_masters;//00 - No master // 01 - Master 1 // 10 - Master
 wire sb_mastlock;//output from arbiter about master lock 1- lock 0-no lock
to all slaves
 wire [31:0] RDATA;//Read data from slave mux
 wire [13:0] HADDR; // Address from Address mux
 wire scl_0;//input to slave 0 about its selection 1 -selected
 wire scl_1;
 wire scl_2;
 wire [1:0] sel_slave;// to Decoder about the slave selected
 wire [31:0] WDATA;//Write data from write data mux\

always #5 clk=~clk;

initial begin
  clk=0;
 rst=1;//reset
  sb_lock_m1=0;//lock request from m1 to arbiter
  req1=1;//request to aquire the bus from Master1
 req2=0;//request to aquire the bus from Master2
  sb_lock_m2=0;//lock request from m2 to arbiter
 resp=0;//Responce from mux
 resp0=0;//responce from slave 0
 resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
 HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
 RDATA_S0=0;//Read Data from Slave 0
 RDATA_S1=1;//Read Data from Slave 1
 RDATA_S2=2;//Read data from Slave 2
 WDATA_M1=1;//Write data from M1
```

```
 WDATA_M2=2;//Write data from M2
 sb_split_ar=0;
 #50

rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=1;//request to aquire the bus from Master1
req2=0;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=0;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=1;//lock request from m1 to arbiter
 req1=1;//request to aquire the bus from Master1
req2=0;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=1;//lock request from m1 to arbiter
 req1=0;//request to aquire the bus from Master1
```

```
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=0;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=1;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=0;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
```

```
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=0;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=3;//Responce from mux
resp0=3;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=0;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=0;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=2;
```

```
#50
rst=0;//reset
 sb_lock_m1=0;//lock request from m1 to arbiter
 req1=1;//request to aquire the bus from Master1
req2=1;//request to aquire the bus from Master2
 sb_lock_m2=0;//lock request from m2 to arbiter
resp=0;//Responce from mux
resp0=0;//responce from slave 0
resp1=0;//responce from slave 1
resp2=0;//responce from slave 2
HADDR_M1=1;//Address of Master 1
HADDR_M2=2;//Address of Master 2
RDATA_S0=0;//Read Data from Slave 0
RDATA_S1=1;//Read Data from Slave 1
RDATA_S2=2;//Read data from Slave 2
WDATA_M1=1;//Write data from M1
WDATA_M2=2;//Write data from M2
sb_split_ar=0;
end
Bus                                          bus_instance(clk,rst,sb_lock_m1
,req1,req2,sb_lock_m2,resp,resp0,resp1,resp2,HADDR_M1,HADDR_M2,RDATA_S0,RDA
TA_S1,RDATA_S2,WDATA_M1,
WDATA_M2,
sb_split_ar,
gnt1,
gnt2,
sb_masters,
sb_mastlock,
RDATA,
HADDR,
scl_0,
scl_1,
scl_2,
sel_slave,
WDATA

);


Endmodule
```

7.2. Two to One Mux for Write Data

```
module Mux_2_1_Write_Data(
in_0      , // Mux first input
in_1      , // Mux Second input
sel        , // Select input
mux_out      // Mux output
    );
    input  [1:0] sel ;
    input [31:0] in_0;
    input [31:0] in_1;
    //-----------Output Ports---------------
    output [31:0] mux_out;
    //------------Internal Variables--------
    reg [31:0]   mux_out;
    //-------------Code Starts Here---------
    always @ (sel or in_0 or in_1)
    begin : MUX
```

```
      if (sel == 1) begin
          mux_out = in_0;
      end else begin
          mux_out = in_1 ;
      end
    end

endmodule
```

7.3. Three to one mux for Address

```
module Mux_3_1( in_0  , // Mux first input
in_1      , // Mux Second input
in_2     ,// Mux Third Input
sel        , // Select input
mux_out      // Mux output
);
//-----------Input Ports---------------
input [1:0] sel ;
input [31:0] in_0;
input [31:0] in_1;
input [31:0] in_2;
//-----------Output Ports---------------
output [31:0] mux_out;
//------------Internal Variables--------
reg [31:0]   mux_out;
//-------------Code Starts Here---------
always @ (sel or in_0 or in_1 or in_2)
begin : MUX
  if (sel == 2'b00) begin
      mux_out = in_0;
  end else if(sel == 2'b01)begin
      mux_out = in_1 ;
      end
      else if(sel == 2'b10)begin
            mux_out = in_2 ;
            end
            else
            mux_out=in_0;//Default when no input given output is slave 0

  end
endmodule
```

7.4. Three to One Mux for response

```
module Mux_3_1_Resp( in_0  , // Mux first input
in_1      , // Mux Second input
in_2     ,// Mux Third Input
sel        , // Select input
mux_out      // Mux output
);
//-----------Input Ports---------------
input [1:0] sel ;
input [1:0] in_0;
input [1:0] in_1;
input [1:0] in_2;
```

```
//-----------Output Ports---------------
output [1:0] mux_out;
//------------Internal Variables--------
reg [1:0]   mux_out;
//-------------Code Starts Here---------
always @ (sel or in_0 or in_1 or in_2)
begin : MUX
  if (sel == 2'b00) begin
      mux_out = in_0;
  end else if(sel == 2'b01)begin
      mux_out = in_1 ;
      end
      else if(sel == 2'b10)begin
            mux_out = in_2 ;
            end
            else
            mux_out=in_0;//Default when no input given output is slave 0

  end
endmodule
```

7.5. Test Case 1 - Verifying the Reset

```
initial begin

clk<=0;
rst<=1;  // initialy apply a reset
#5 rst<=0;
//---------------------------------------
req1<=1;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=0; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b01; // OKAY resonse from the slave
sb_ready_ar<=1'b0; // ready signal from the slave
#5
//-----------------------------------
clk<=0;
rst<=1;  // initialy apply a reset
#5 rst<=0;
//-----------------------------------
req1<=01;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=0; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b00; // OKAY resonse from the slave
sb_ready_ar<=1'b1; // ready signal from the slave
#5
//------------------
clk<=0;
rst<=1;  // initialy apply a reset
#5 rst<=0;
//---------------------
req1<=01;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=0; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
```

```
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b11; // OKAY resonse from the slave
sb_ready_ar<=1'b0; // ready signal from the slave
end
//$finish
```

7.6. Test Case 2 - One Master Request the Bus

```
initial begin
clk<=0;
rst<=1;  // initialy apply a reset
#5 rst<=0;
//--------------------------------------
req1<=0;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=1; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b00; // OKAY resonse from the slave
sb_ready_ar<=1'b1; // ready signal from the slave
#5
//----------------------------------
req1<=1;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=0; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b00; // OKAY resonse from the slave
sb_ready_ar<=1'b1; // ready signal from the slave
end
//$finish
```

7.7. Test Case 3 - Both Masters Request the Bus

```
initial begin
clk<=0;
rst<=1;  // initialy apply a reset
#5 rst<=0;
//--------------------------------------
req1<=1;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=1; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b00; // OKAY resonse from the slave
sb_ready_ar<=1'b0; // ready signal from the slave
#5
//----------------------------------
req1<=0;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=1; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b00; // OKAY resonse from the slave
sb_ready_ar<=1'b1; // ready signal from the slave
end
//$finish
```

7.8. Test Case 4 - Split Transaction

```
initial begin
clk<=0;
rst<=1;  // initialy apply a reset
#5 rst<=0;
//--------------------------------------
req1<=1;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=0; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b01; // OKAY resonse from the slave
sb_ready_ar<=1'b0; // ready signal from the slave
#5
//-----------------------------------
req1<=01;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=0; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b00; // OKAY resonse from the slave
sb_ready_ar<=1'b1; // ready signal from the slave
#5
req1<=01;  //master1 reqest for the bus
sb_lock_m1<=0; // unlocking transfer
req2<=0; //master 2 also reqest for the bus
sb_lock_m2<=0; // unlocking transfer
sb_split_ar<=2'b00;  // no split signal from any slave
sb_resp_ar<=2'b11; // OKAY resonse from the slave
sb_ready_ar<=1'b0; // ready signal from the slave
end
//$finish
```