Assignment –3 Part_A

VLSI Design Lab

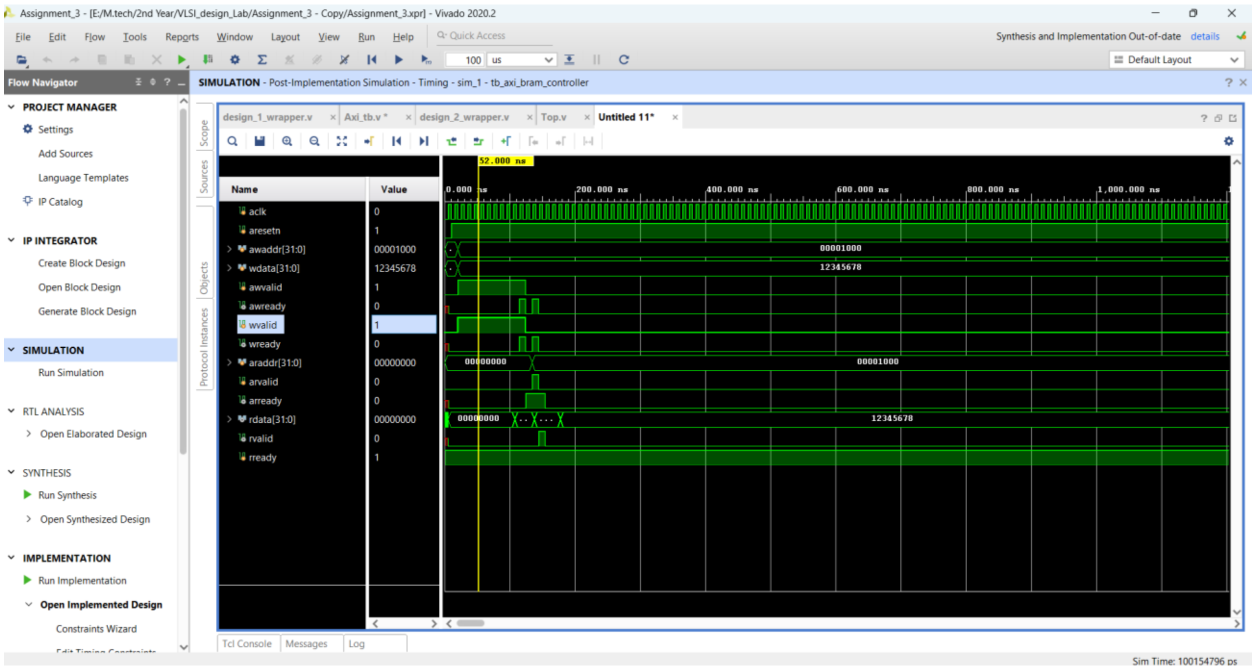
Manish Ranjan

24M1176

1.

Post Implementation Timing Simulation:

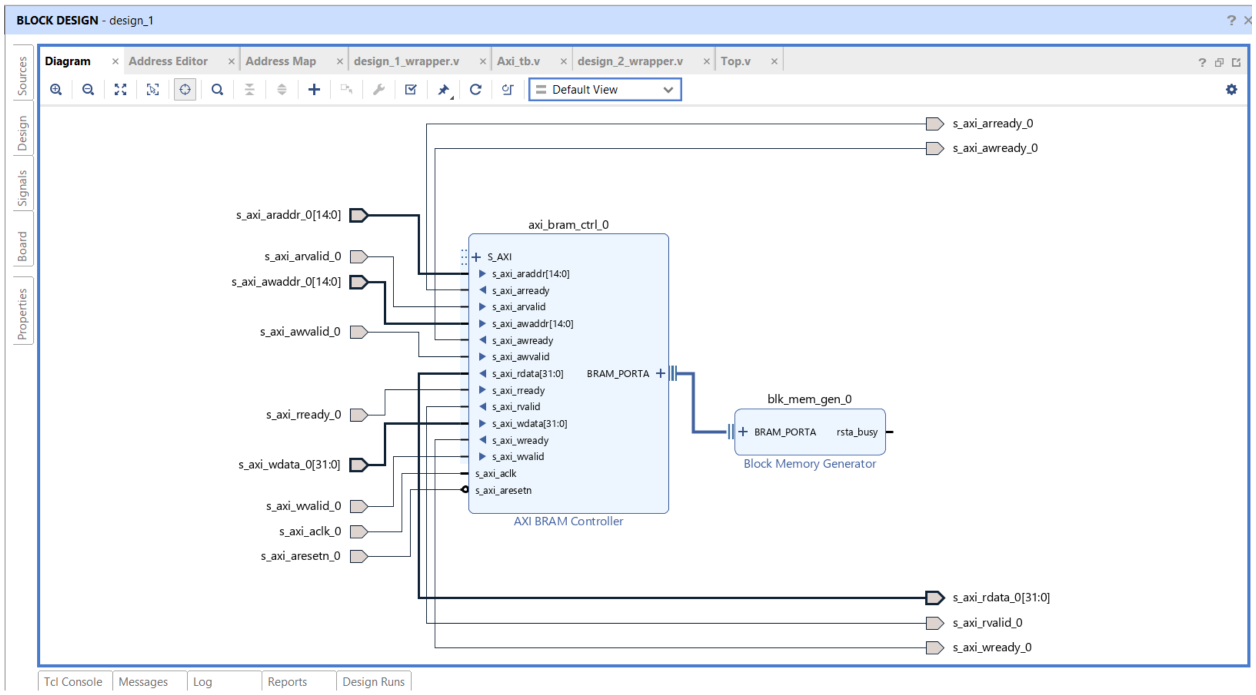
Writing 12345678 at 00001000 and Reading it.



SIMULATION - Post-Implementation Simulation - Timing - sim_1 - tb_axi_bram_controller

Name	Value
aclk	0
aresetn	1
awaddr[31:0]	00001000
wdata[31:0]	12345678
awvalid	1
awready	0
wvalid	1
wready	0
araddr[31:0]	00000000
arvalid	0
arready	0
rdata[31:0]	00000000
rvalid	0
rready	1

Block design diagram:



Testbench(Master):

`timescale 1ns / 1ps

module Axi_tb;

// Declare AXI signals reg aclk;

reg aresetn;

// AXI signal reg [31:0] awaddr; // Write address reg [31:0] wdata; // Write data

reg awvalid; // Write address valid reg wvalid; // Write valid wire awready; // Write address ready wire wready; // Write data ready

reg [31:0] araddr; // Read address reg arvalid; // Read address valid wire arready; // Read address ready wire [31:0] rdata; // Read data wire rvalid; // Read valid reg rready; // Read ready

design_1_wrapper dut (.s_axi_aclk_0(aclk), .s_axi_aresetn_0(aresetn), .s_axi_araddr_0(araddr), .s_axi_arready_0(arready), .s_axi_arvalid_0(arvalid), .s_axi_awaddr_0(awaddr), .s_axi_awready_0(awready), . s_axi_awvalid_0(awvalid), .s_axi_rdata_0(rdata), .s_axi_rready_0(rready), .s_axi_rvalid_0(rv alid), .s_axi_wdata_0(wdata), .s_axi_wready_0(wready), .s_axi_wvalid_0(wvalid));

always #5 aclk = ~aclk;

initial begin

```
aclk = 0;
aresetn = 0;
awaddr = 32'h0;
awvalid = 0;
wvalid = 0;
araddr = 32'h0;
arvalid = 0;
rready = 1;


#10 aresetn = 1;

// Write operation

#10 awaddr = 32'h0000_1000;
    wdata = 32'h12345678;
    awvalid = 1;
    wvalid = 1;


wait(awready && wready);
#10 awvalid = 0;
    wvalid = 0;


#10 araddr = 32'h0000_1000; // Same address as the write
    arvalid = 1;


wait(arready);
#10 arvalid = 0;


wait(rvalid);
```

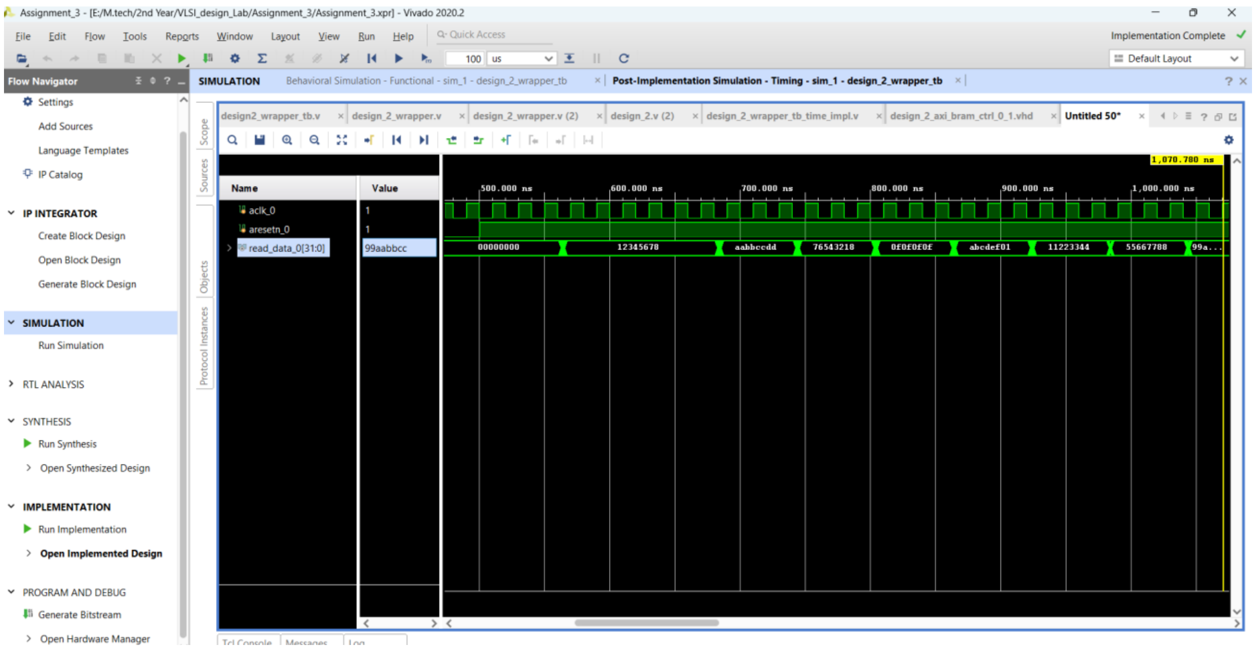
```
#10 $finish;

end

endmodule
```

2. Post Implementation Timing Simulation :

Matched with Bram coe file given below the output



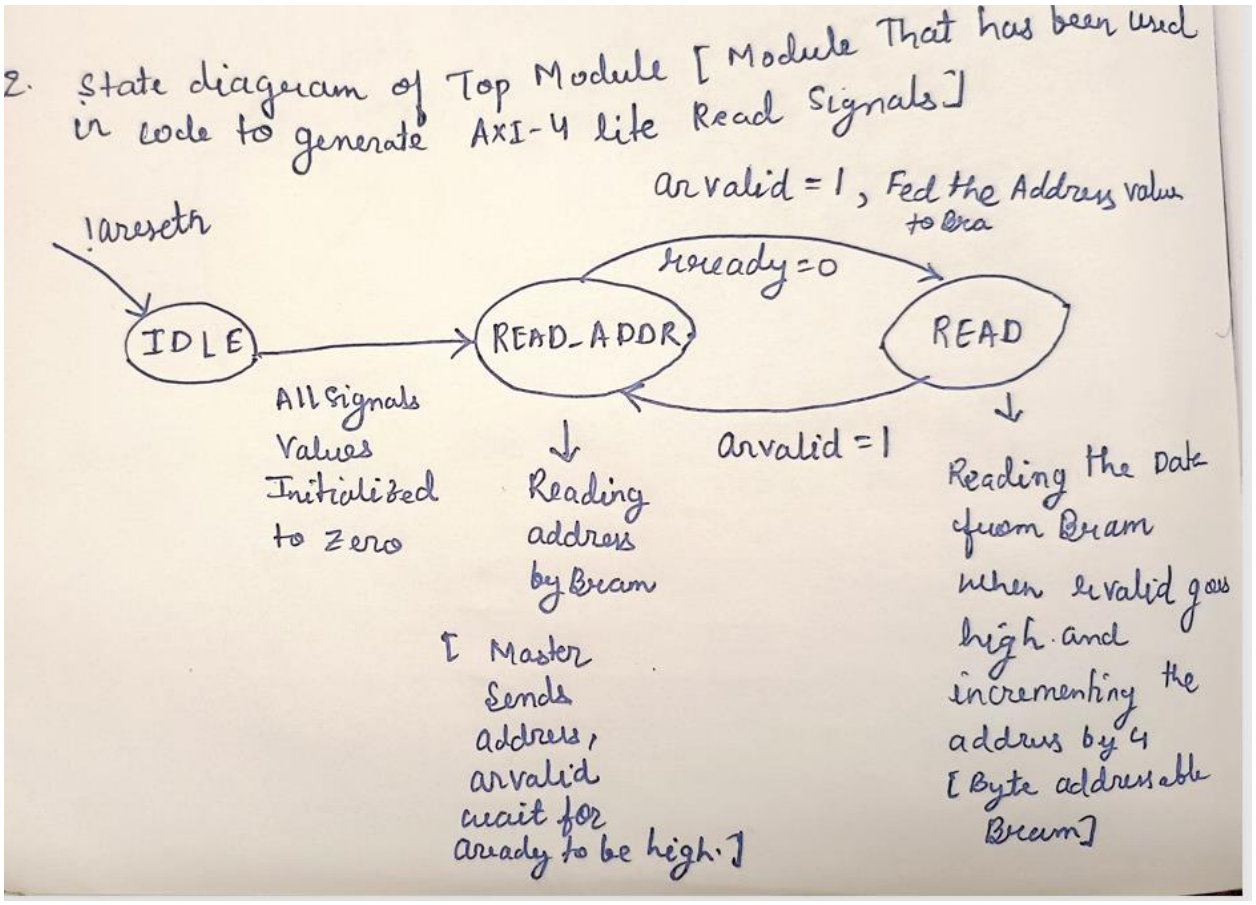
```
Name                Value
aclk_0              1
aresetn_0           1
read_data_0[31:0]   99aabbcc

read_data_0[31:0]: 00000000 | 12345678 | aabbccdd | 76543218 | 0f0f0f0f | abcdef01 | 11223344 | 55667788 | 99a...
```

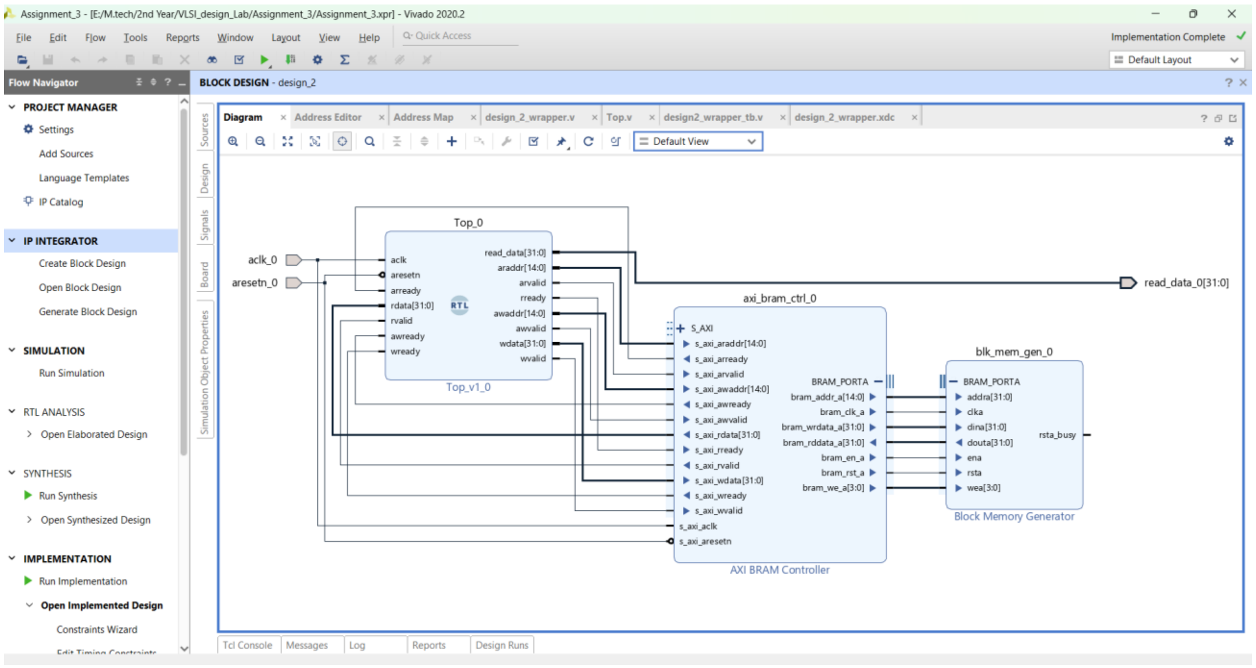
Bram Coe file content:

memory_initialization_radix=16; memory_initialization_vector=12345678 AABBCCDD 76543218 0F0F0F0F ABCDEF01 11223344 55667788 99AABBCC DAEEFF00 CFFEBBBE;

State Diagram:


2. State diagram of Top Module [Module That has been used in code to generate AXI-4 lite Read Signals]
!aresetn
IDLE
READ_ADDR
READ
arvalid = 1, Fed the Address value to Bra
rready = 0
arvalid = 1
All Signals Values Initialized to Zero
Reading address by Bram
[Master Sends address, arvalid wait for arready to be high.]
Reading the Data from Bram when rvalid goes high and incrementing the address by 4 [Byte addressable Bram]

Block design Diagram :



Verilog code and Testbench :

Top Module code used for generating AXI4 lite Read Signals:

```
`timescale 1ns / 1ps

module Top (

input wire aclk,
input wire aresetn,
output  [31:0] read_data,

// AXI Read Address Channel
output reg [14:0] araddr,
input wire arready,
output reg arvalid,

// AXI Read Data Channel
input wire [31:0] rdata,
input wire rvalid,
output reg rready,

// AXI Write Address Channel
output [14:0] awaddr,
input awready,
output reg awvalid,

// AXI Write Data Channel
output [31:0] wdata,
input wready,
output reg wvalid


);

reg [1:0] state, next_state;
parameter IDLE = 2'b00, READ_ADDR = 2'b01, READ = 2'b10;



// Sequential Logic for State Transitions
always @(posedge aclk or negedge aresetn) begin
    if (!aresetn)
        state <= IDLE;
    else
        state <= next_state;
end

//sequential logic for signal Assignments for Read
always @(posedge aclk) begin

        case (state)
            IDLE: begin
                araddr  <= 15'h0000;
                arvalid <= 0;
                rready  <= 0;
                wvalid <=0;
                awvalid<=0;

            end

            READ_ADDR: begin
```

```
                wvalid <=0;
                awvalid<=0;
                arvalid <= 1;
                rready  <= 0;
            end

            READ: begin
                wvalid <=0;
                awvalid<=0;
                arvalid <= 0;
                rready  <= 1;
                if (rvalid) begin

                    if (araddr > 36)
                        araddr <= 0;
                    else
                        araddr <= araddr + 4;

               end
               else
                araddr <=araddr;
            end
        endcase
    end


// Combinational Logic for Next State Decisions
always @(*) begin
    next_state = state;

    case (state)
        IDLE: begin
            next_state = READ_ADDR;
        end

        READ_ADDR: begin
            if (arready)
                next_state = READ;
                else
                next_state = READ_ADDR;
        end

        READ: begin
            if (rvalid)
                next_state = READ_ADDR;
                else
                next_state = READ;
        end
    endcase
end
//output logic
assign read_data= rdata;
```

endmodule

Design wrapper code:

`timescale 1 ps / 1 ps

module design_2_wrapper (aclk_0, aresetn_0, read_data_0); input aclk_0; input aresetn_0; output [31:0]read_data_0;

wire aclk_0; wire aresetn_0; wire [31:0]read_data_0;

design_2 design_2_i (.aclk_0(aclk_0), .aresetn_0(aresetn_0), .read_data_0(read_data_0)); endmodule

Testbench :

module design_2_wrapper_tb;

reg aclk_0; reg aresetn_0; wire [31:0]read_data_0;

// Instantiate the design wrapper design_2_wrapper uut
(.aclk_0(aclk_0), .aresetn_0(aresetn_0), .read_data_0(read_data_0));

always begin #10 aclk_0 = ~aclk_0;

 end

initial begin

```
aclk_0 = 0;
aresetn_0 = 0;


#300;
aresetn_0 = 1;


#2500;
$finish;
```

end

// Monitor signals initial begin $monitor("Time = %0t | aclk_0 = %b | aresetn_0 = %b", $time, aclk_0, aresetn_0); end

endmodule