Laboratory Exercise #3
Storage Elements in Verilog

Objective

The third laboratory's objective is to design and simulate a 32-by-32 register file and a simple data memory module to understand different types of storage elements that are used in the MIPS microarchitecture. Within the 32-by-32 register file is an array of 32 different 32-bit registers constructed from flip-flops or, in the case of a larger processor or an FPGA implementation, SRAM cells and uses multiplexers and decoders to provide a means for reading and writing. With much larger size, comes data memory that provides the microprocessor with a means of storing more long-term data.

Design

1. Launch and install vivado and create a new design project.

a) Open a terminal window and create a home user directory to save lab projects. Run commands to install and launch vivado.
b) Create a new project called 'lab3' and Leave the device properties as default.

2. Design and simulate a 32-by-32 register file using behavioural Verilog

a) Add sources for simulating a 32-by-32 register file using behavioural Verilog.
b) Implement design source code logic for 32-by-32 register file top block module by executing statements sequentially.
c) Implement stimulation/testbench code for testing the simulation results of the 32-by-32 register file by varying reset and clock signal cycle time that drives the design block for short time intervals.
d) Run synthesis and implementation post which run Behavioural stimulation of the code logic and observe results on waveform generator.
e) Synthesize the implementation with the following device properties:
   Device Family: Virtex5
   Device: XC5VLX110T
   Package: ff1136
   Speed Grade: -1
f) Monitor the outputs and capture the results obtained.

3. Source Files:

a) ****BEHAVIORAL VERILOG FOR 32-BY-32 REGISTER FILE****

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/18/2019 07:51:44 AM
// Design Name:
// Module Name: rf
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module rf(BusA, BusB, BusW, RA,                          //Module initialization with input and output params
RB, RW, RegWr, Clk);

output [31:0] BusA;                                      //Defining 32 bit output variable for read outputA
output [31:0] BusB;                                      //Defining 32 bit output variable for read outputB

input [31:0] BusW;                                       //Defining 32 bit input variable for 32 bit write input
input [4:0] RA , RB, RW;                                 //Defining 5 bit input variable for read outputB

input RegWr;                                             //write enable on negative edge clock cycle
input Clk;

reg [31:0] registers [31:0];

   always @ (negedge Clk) begin          //To perform write operation on the negative edge of the clock
       if(RegWr && RW!= 5'd0) begin       // Write into RW when the 0th register is wired to 0 & RegWr is
                                          enabled high
       registers [RW] <= BusW;
      end
    end

    assign BusA = (RA == 5'd0)?0: registers[RA] ;      //assign value provided on the 5 line registers RA
                                                        & RB to 32 bit BusA and BusB
    assign BusB = (RB == 5'd0)?0: registers[RB] ;
endmodule
```

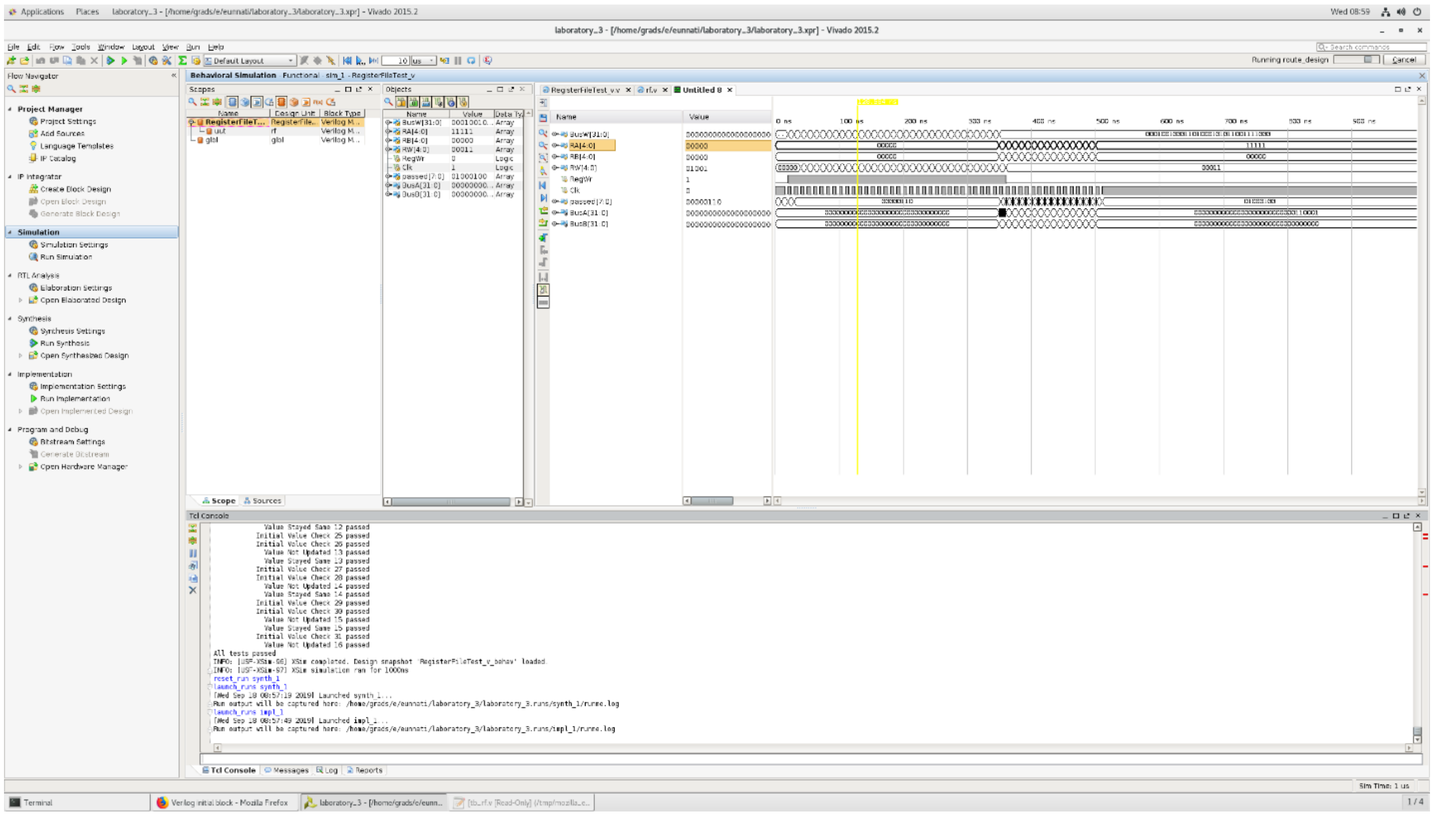
b) Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   09:05:47 09/26/2017
// Design Name:   rf
// Module Name:   /home/grads/y/yc2704/ecen651/lab3/tb_rf.v
// Project Name:  lab3
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: rf
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

`define STRLEN 32
module RegisterFileTest_v;
task passTest;
input [31:0] actualOut, expectedOut;
input [`STRLEN*8:0] testType;
inout [7:0] passed;
if(actualOut == expectedOut) begin $display ("%s passed", testType); passed = passed + 1; end
else $display ("%s failed: %d should be %d", testType, actualOut, expectedOut);
endtask
task allPassed;
input [7:0] passed;
input [7:0] numTests;
if(passed == numTests) $display ("All tests passed");
else $display("Some tests failed");
endtask

// Inputs
reg [31:0] BusW;
reg [4:0] RA;
reg [4:0] RB;
reg [4:0] RW;
reg RegWr;
reg Clk;
reg [7:0] passed;

// Outputs
wire [31:0] BusA;
wire [31:0] BusB;

// Instantiate the Unit Under Test (UUT)
rf uut (
.BusA(BusA),
.BusB(BusB),
.BusW(BusW),
.RA(RA),
.RB(RB),
.RW(RW),
.RegWr(RegWr),
.Clk(Clk)
);
initial begin
// Initialize Inputs
BusW = 0;
RA = 0;
RB = 0;
RW = 0;
RegWr = 0;
Clk = 1;
passed = 0;

#10;
// Add stimulus here
```

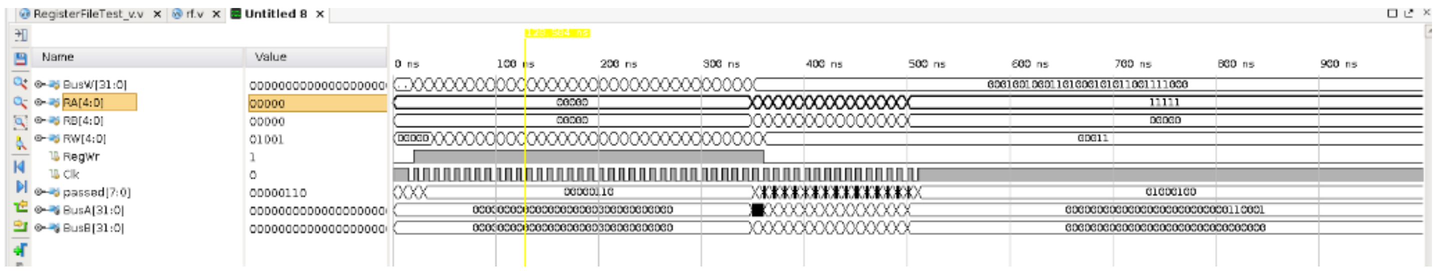
```
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd0, 32'h0, 1'b0};
passTest(BusA, 32'h0, "Initial $0 Check 1", passed);
passTest(BusB, 32'h0, "Initial $0 Check 2", passed);
#5; Clk = 0; #5; Clk = 1;

{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd0, 32'h12345678, 1'b1};
passTest(BusA, 32'h0, "Initial $0 Check 3", passed);
passTest(BusB, 32'h0, "Initial $0 Check 4", passed);
#5; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h0, "$0 Stays 0 Check 1", passed);
passTest(BusB, 32'h0, "$0 Stays 0 Check 2", passed);

{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd0, 32'h0, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd1, 32'h1, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd2, 32'h2, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd3, 32'h3, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd4, 32'h4, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd5, 32'h5, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd6, 32'h6, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd7, 32'h7, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd8, 32'h8, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd9, 32'h9, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd10, 32'h10, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd11, 32'h11, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd12, 32'h12, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd13, 32'h13, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd14, 32'h14, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd15, 32'h15, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd16, 32'h16, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd17, 32'h17, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd18, 32'h18, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd19, 32'h19, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd20, 32'h20, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd21, 32'h21, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd22, 32'h22, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd23, 32'h23, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd24, 32'h24, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd25, 32'h25, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd26, 32'h26, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd27, 32'h27, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd28, 32'h28, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd29, 32'h29, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd30, 32'h30, 1'b1};#5; Clk = 0; #5; Clk = 1;
{RA, RB, RW, BusW, RegWr} = {5'd0, 5'd0, 5'd31, 32'h31, 1'b1};#5; Clk = 0; #5; Clk = 1;

{RA, RB, RW, BusW, RegWr} = {5'd1, 5'd2, 5'd1, 32'h12345678, 1'b1};
#2;
passTest(BusA, 32'h1, "Initial Value Check 1", passed);
passTest(BusB, 32'h2, "Initial Value Check 2", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h12345678, "Value Updated 1", passed);
passTest(BusB, 32'h2, "Value Stayed Same 1", passed);

{RA, RB, RW, BusW, RegWr} = {5'd3, 5'd4, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h3, "Initial Value Check 3", passed);
passTest(BusB, 32'h4, "Initial Value Check 4", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h3, "Value Not Updated 2", passed);
passTest(BusB, 32'h4, "Value Stayed Same 2", passed);

{RA, RB, RW, BusW, RegWr} = {5'd5, 5'd6, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h5, "Initial Value Check 5", passed);
passTest(BusB, 32'h6, "Initial Value Check 6", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h5, "Value Not Updated 3", passed);
passTest(BusB, 32'h6, "Value Stayed Same 3", passed);

{RA, RB, RW, BusW, RegWr} = {5'd7, 5'd8, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h7, "Initial Value Check 7", passed);
passTest(BusB, 32'h8, "Initial Value Check 8", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h7, "Value Not Updated 4", passed);
passTest(BusB, 32'h8, "Value Stayed Same 4", passed);

{RA, RB, RW, BusW, RegWr} = {5'd9, 5'd10, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h9, "Initial Value Check 9", passed);
passTest(BusB, 32'h10, "Initial Value Check 10", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h9, "Value Not Updated 5", passed);
passTest(BusB, 32'h10, "Value Stayed Same 5", passed);

{RA, RB, RW, BusW, RegWr} = {5'd11, 5'd12, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h11, "Initial Value Check 11", passed);
passTest(BusB, 32'h12, "Initial Value Check 12", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h11, "Value Not Updated 6", passed);
passTest(BusB, 32'h12, "Value Stayed Same 6", passed);

{RA, RB, RW, BusW, RegWr} = {5'd13, 5'd14, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h13, "Initial Value Check 13", passed);
passTest(BusB, 32'h14, "Initial Value Check 14", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h13, "Value Not Updated 7", passed);
passTest(BusB, 32'h14, "Value Stayed Same 7", passed);

{RA, RB, RW, BusW, RegWr} = {5'd15, 5'd16, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h15, "Initial Value Check 15", passed);
passTest(BusB, 32'h16, "Initial Value Check 16", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h15, "Value Not Updated 8", passed);
passTest(BusB, 32'h16, "Value Stayed Same 8", passed);

{RA, RB, RW, BusW, RegWr} = {5'd17, 5'd18, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h17, "Initial Value Check 17", passed);
passTest(BusB, 32'h18, "Initial Value Check 18", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h17, "Value Not Updated 9", passed);
passTest(BusB, 32'h18, "Value Stayed Same 9", passed);

{RA, RB, RW, BusW, RegWr} = {5'd19, 5'd20, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h19, "Initial Value Check 19", passed);
passTest(BusB, 32'h20, "Initial Value Check 20", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h19, "Value Not Updated 10", passed);
passTest(BusB, 32'h20, "Value Stayed Same 10", passed);

{RA, RB, RW, BusW, RegWr} = {5'd21, 5'd22, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h21, "Initial Value Check 21", passed);
passTest(BusB, 32'h22, "Initial Value Check 22", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h21, "Value Not Updated 11", passed);
passTest(BusB, 32'h22, "Value Stayed Same 11", passed);

{RA, RB, RW, BusW, RegWr} = {5'd23, 5'd24, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h23, "Initial Value Check 23", passed);
passTest(BusB, 32'h24, "Initial Value Check 24", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h23, "Value Not Updated 12", passed);
passTest(BusB, 32'h24, "Value Stayed Same 12", passed);

{RA, RB, RW, BusW, RegWr} = {5'd25, 5'd26, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h25, "Initial Value Check 25", passed);
passTest(BusB, 32'h26, "Initial Value Check 26", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h25, "Value Not Updated 13", passed);
passTest(BusB, 32'h26, "Value Stayed Same 13", passed);

{RA, RB, RW, BusW, RegWr} = {5'd27, 5'd28, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h27, "Initial Value Check 27", passed);
passTest(BusB, 32'h28, "Initial Value Check 28", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h27, "Value Not Updated 14", passed);
passTest(BusB, 32'h28, "Value Stayed Same 14", passed);

{RA, RB, RW, BusW, RegWr} = {5'd29, 5'd30, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h29, "Initial Value Check 29", passed);
passTest(BusB, 32'h30, "Initial Value Check 30", passed);
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h29, "Value Not Updated 15", passed);
passTest(BusB, 32'h30, "Value Stayed Same 15", passed);

{RA, RB, RW, BusW, RegWr} = {5'd31, 5'd32, 5'd3, 32'h12345678, 1'b0};
#2;
passTest(BusA, 32'h31, "Initial Value Check 31", passed);
```

```
#3; Clk = 0; #5; Clk = 1;
passTest(BusA, 32'h31, "Value Not Updated 16", passed);

allPassed(passed, 68);
end

endmodule
```

4) <u>Waveform Generated</u>



```
      Value Stayed Same 12 passed
    Initial Value Check 25 passed
    Initial Value Check 26 passed
      Value Not Updated 13 passed
      Value Stayed Same 13 passed
    Initial Value Check 27 passed
    Initial Value Check 28 passed
      Value Not Updated 14 passed
      Value Stayed Same 14 passed
    Initial Value Check 29 passed
    Initial Value Check 30 passed
      Value Not Updated 15 passed
      Value Stayed Same 15 passed
    Initial Value Check 31 passed
      Value Not Updated 16 passed
All tests passed
INFO: [USF-XSim-96] XSim completed. Design snapshot 'RegisterFileTest_v_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
reset_run synth_1
launch_runs synth_1
[Wed Sep 18 08:57:19 2019] Launched synth_1...
Run output will be captured here: /home/grads/e/eunnati/laboratory_3/laboratory_3.runs/synth_1/runme.log
launch_runs impl_1
[Wed Sep 18 08:57:49 2019] Launched impl_1...
Run output will be captured here: /home/grads/e/eunnati/laboratory_3/laboratory_3.runs/impl_1/runme.log
```



5) <u>Simulate a simple data memory module using Behavioural Structure</u>

a) Add sources for simulating a data memory module using behavioural Verilog.
b) Implement design source code logic for data memory module top block module by executing statements sequentially.
c) Implement stimulation/testbench code for testing the simulation results of the data memory module by varying reset and clock signal cycle time that drives the design block for short time intervals.
d) Run synthesis and implementation post which run Behavioural stimulation of the code logic and observe results on waveform generator.
e) Monitor the outputs and capture the results obtained.

6) Source Files:

a) ****Data Memory Module using Behavioural****

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/18/2019 09:08:30 AM
// Design Name:
// Module Name: DataMemory
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module DataMemory (ReadData, Address, WriteData                    //Module initialization with input and output params
            , MemoryRead, MemoryWrite, Clock);

    output reg [31:0] ReadData;                                     //Defining 32 bit output variable for read output
    input wire [31:0] WriteData;                                    //Defining 32 bit input variable for write data
    input wire [5:0] Address ;                                      // Defining 5 bit input for address

    input wire MemoryRead, MemoryWrite ;                    //read and write enable on positive/negative edge clock cycle
    input wire Clock;

    reg [31:0] regs [63:0];


  always @ (negedge Clock) begin             //Writing data when MemoryWrite goes high on the negative edge of the clock
    if(MemoryWrite) begin
       regs[Address] <= WriteData;
       end
   end

   always @ (posedge Clock) begin           //Reading data when MemoryRead goes high on the positive edge of the clock
       if(MemoryRead) begin
       ReadData = regs[Address];
          //regs[Address] <= WriteData;
          end
      end
endmodule
```

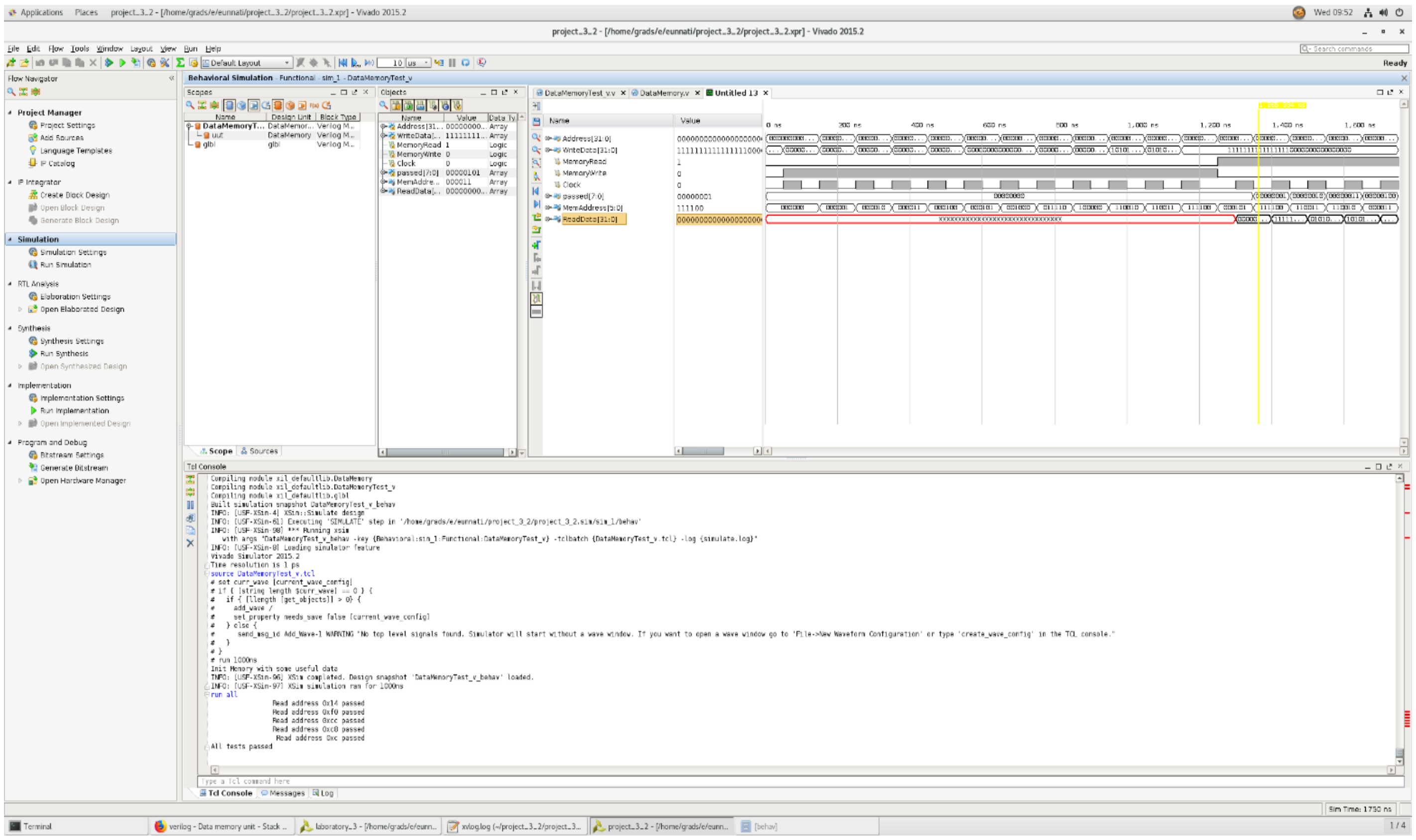
8. Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   09:44:35 09/26/2017
// Design Name:   DataMemory
// Module Name:   /home/grads/y/yc2704/ecen651/lab3_2/tb_DataMemory.v
// Project Name:  lab3_2
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: DataMemory
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
`define STRLEN 32
module DataMemoryTest_v;


task passTest;
input [31:0] actualOut, expectedOut;
input [`STRLEN*8:0] testType;
inout [7:0] passed;

if(actualOut == expectedOut) begin $display ("%s passed", testType); passed = passed + 1; end
else $display ("%s failed: %d should be %d", testType, actualOut, expectedOut);
endtask

task allPassed;
input [7:0] passed;
input [7:0] numTests;

if(passed == numTests) $display ("All tests passed");
else $display("Some tests failed");
endtask


// Inputs
```

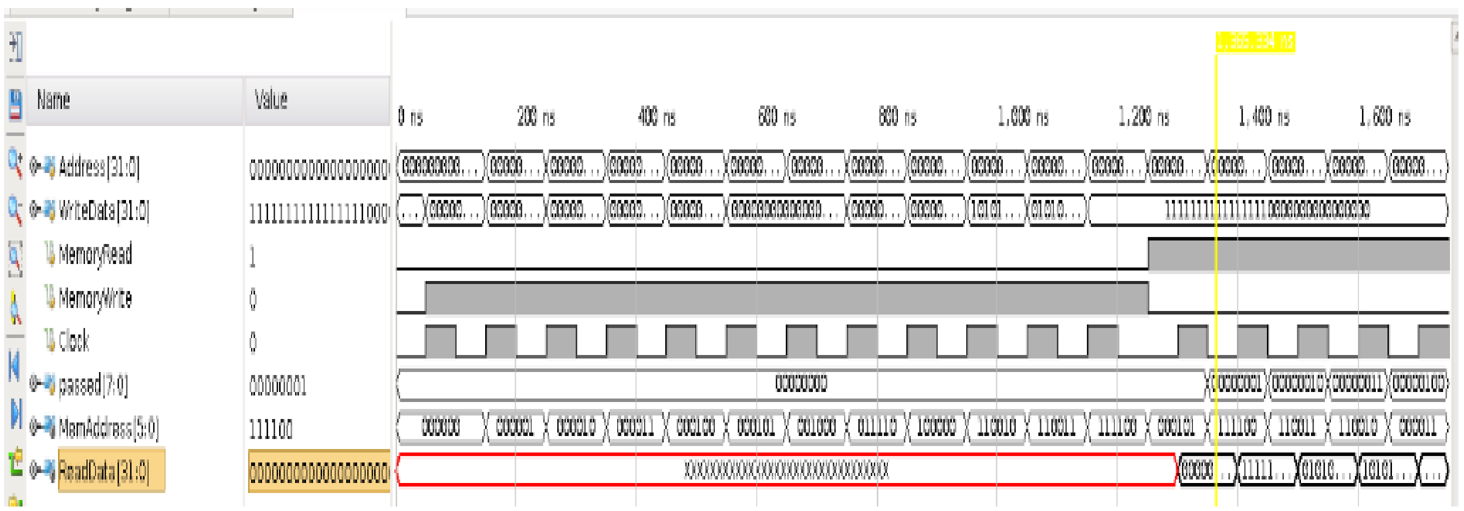
```
reg [31:0] Address;
reg [31:0] WriteData;
reg MemoryRead;
reg MemoryWrite;
reg Clock;
reg [7:0] passed;

   //intermediate nets
wire [5:0] MemAddress;

// Outputs
wire [31:0] ReadData;


assign MemAddress = Address[7:2];

// Instantiate the Unit Under Test (UUT)
DataMemory uut (
.ReadData(ReadData),
.Address(MemAddress),
.WriteData(WriteData),
.MemoryRead(MemoryRead),
.MemoryWrite(MemoryWrite),
.Clock(Clock)
);

initial begin
// Initialize Inputs
Address = 0;
WriteData = 0;
MemoryRead = 0;
MemoryWrite = 0;
Clock = 0;
passed = 0;


// Add stimulus here
$display("Init Memory with some useful data");
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h0, 32'h4, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h4, 32'h3, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h8, 32'd50, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'hc, 32'd40, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h10, 32'd30, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h14, 32'h0, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h20, 32'h0, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h78, 32'h132, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'h80, 32'd16435934, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'hc8, 32'haaaaffff, 2'h2};#50 Clock = 0;
```

```
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'hcc, 32'd1431699200, 2'h2};#50 Clock = 0;
#50 Clock = 1;{Address, WriteData, MemoryWrite, MemoryRead} = {32'hf0, 32'hffff0000, 2'h2};#50 Clock = 0;

#50 Clock = 0;

{Address, WriteData, MemoryWrite, MemoryRead} = {32'h14, 32'hffff0000, 2'h1};
#50 Clock = 1;
#50 Clock = 0;
passTest(ReadData, 32'h0, "Read address 0x14", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hf0, 32'hffff0000, 2'h1};
#50 Clock = 1;
#50 Clock = 0;
passTest(ReadData, 32'hffff0000, "Read address 0xf0", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hcc, 32'hffff0000, 2'h1};
#50 Clock = 1;
#50 Clock = 0;
passTest(ReadData, 32'd1431699200, "Read address 0xcc", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hc8, 32'hffff0000, 2'h1};
#50 Clock = 1;
#50 Clock = 0;
passTest(ReadData, 32'haaaaffff, "Read address 0xc8", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hc, 32'hffff0000, 2'h1};
#50 Clock = 1;
#50 Clock = 0;
passTest(ReadData, 32'd40, "Read address 0xc", passed);

allPassed(passed, 5);

end

endmodule
```

7) Waveform Generated

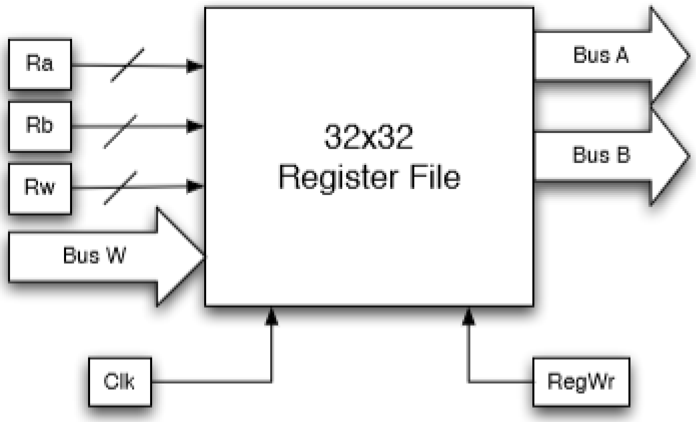


```
Compiling module xil_defaultlib.DataMemory
Compiling module xil_defaultlib.DataMemoryTest_v
Compiling module xil_defaultlib.glbl
Built simulation snapshot DataMemoryTest_v_behav
INFO: [USF-XSim-4] XSim::Simulate design
INFO: [USF-XSim-61] Executing 'SIMULATE' step in '/home/grads/e/eunnati/project_3_2/project_3_2.sim/sim_1/behav'
INFO: [USF-XSim-98] *** Running xsim
   with args "DataMemoryTest_v_behav -key {Behavioral:sim_1:Functional:DataMemoryTest_v} -tclbatch {DataMemoryTest_v.tcl} -log {simulate.log}"
INFO: [USF-XSim-8] Loading simulator feature
Vivado Simulator 2015.2
Time resolution is 1 ps
source DataMemoryTest_v.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Waveform Configuration' or type 'create_wave_config' in the TCL console."
#   }
# }
# run 1000ns
Init Memory with some useful data
INFO: [USF-XSim-96] XSim completed. Design snapshot 'DataMemoryTest_v_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
run all
                Read address 0x14 passed
                Read address 0xf0 passed
                Read address 0xcc passed
                Read address 0xc0 passed
                 Read address 0xc passed
All tests passed
```

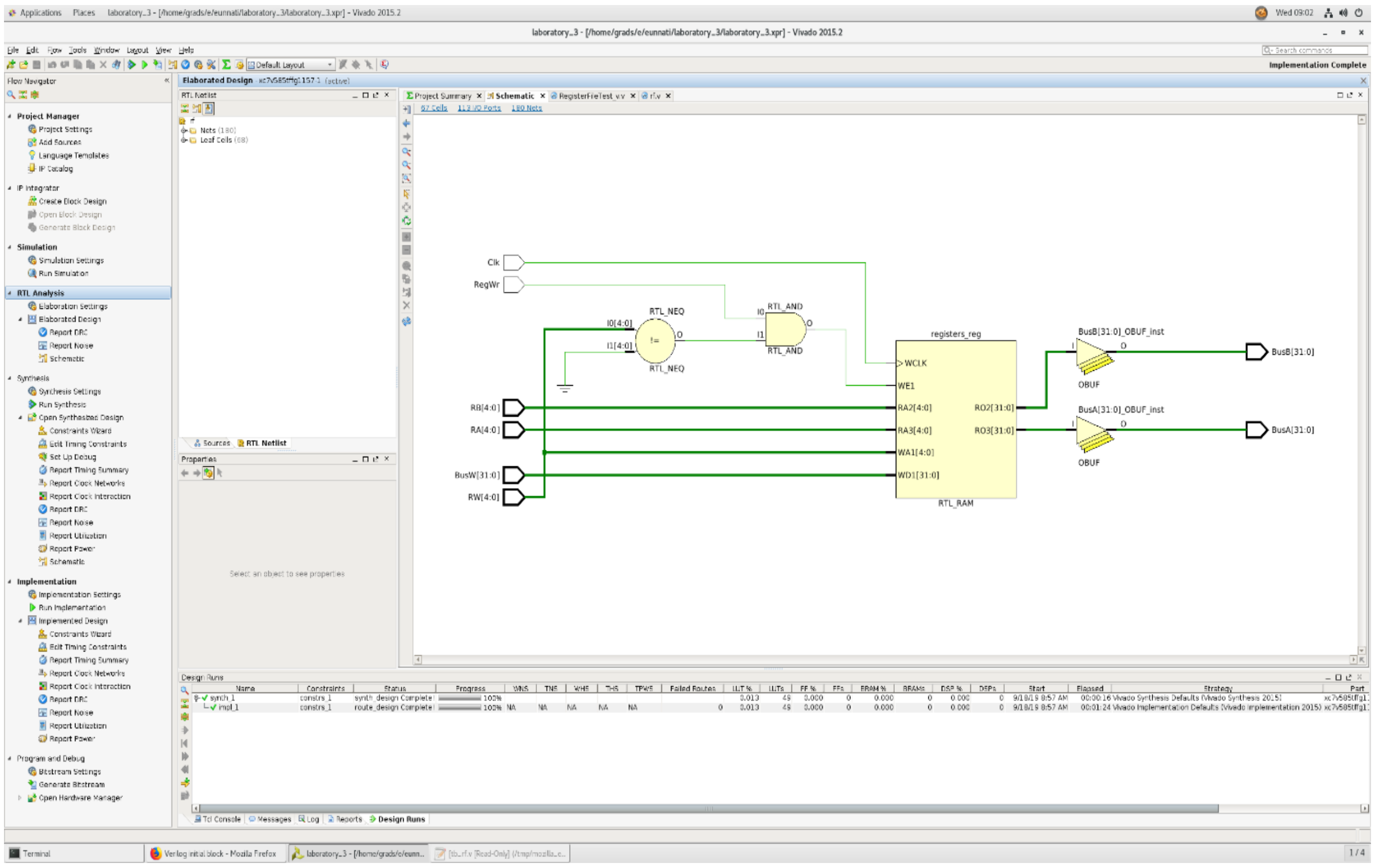


SCHEMATICS

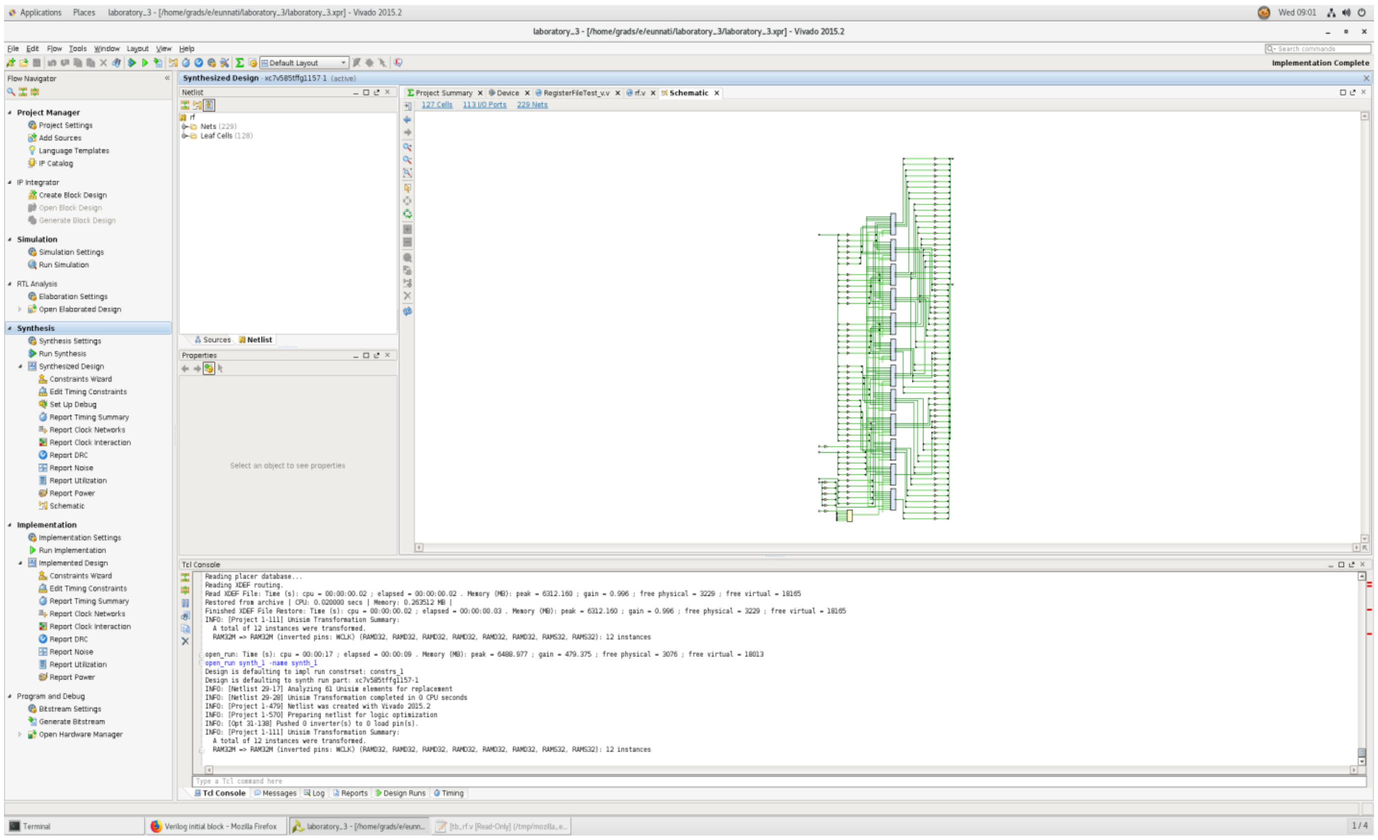
1. 32-by-32 Register File



The port interface for the MIPS Register File



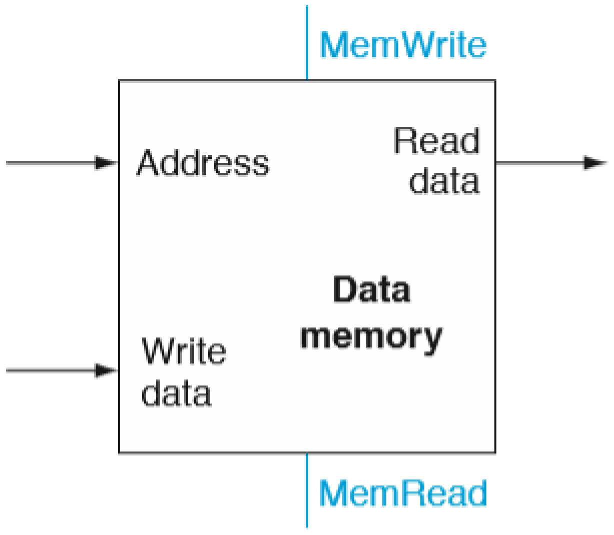
Schematic of MIPS Register File following Behavioural Verilog



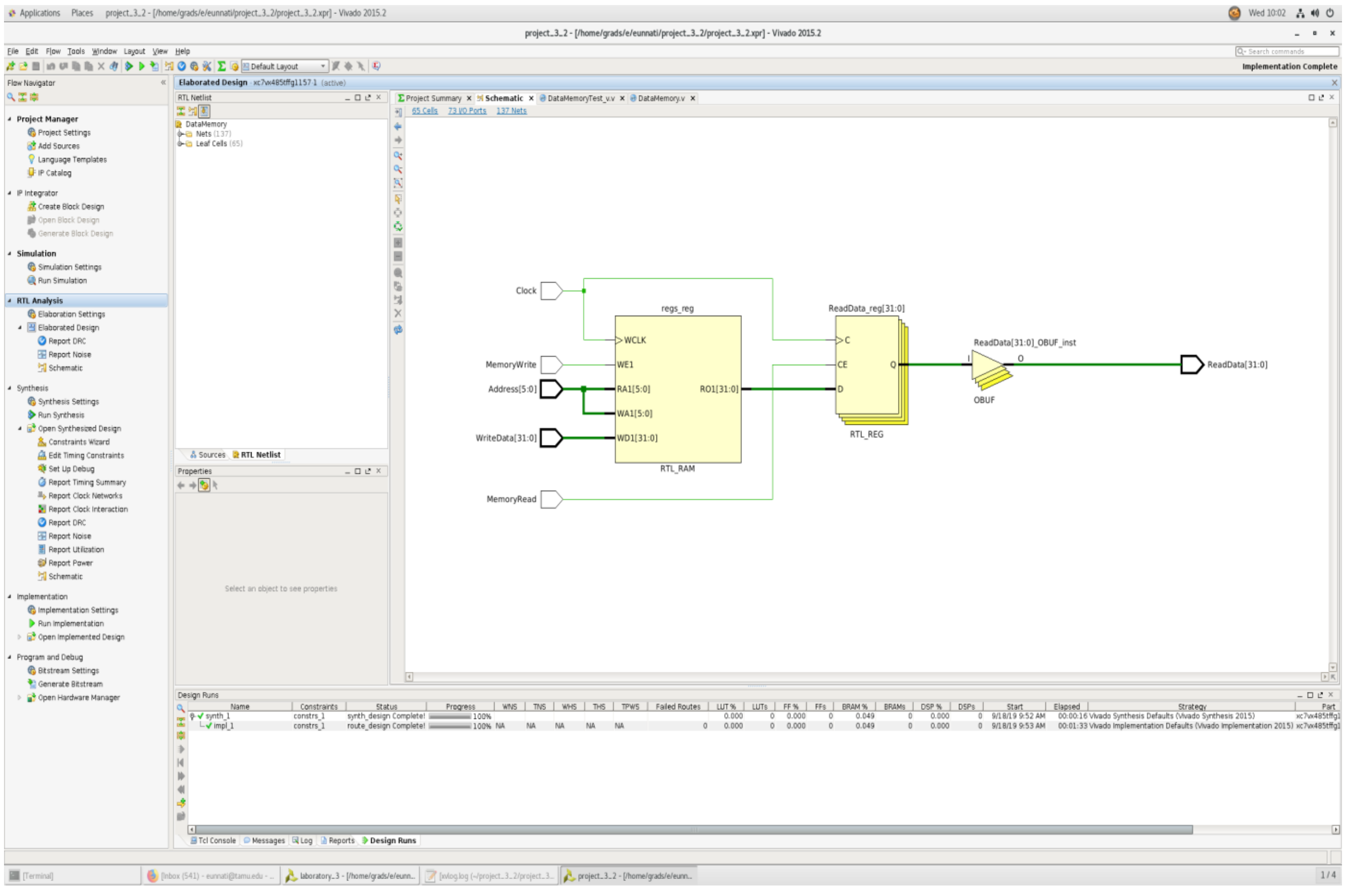
```
Reading placer database...
Reading XDEF routing.
Read XDEF File: Time (s): cpu = 00:00:00.02 ; elapsed = 00:00:00.02 . Memory (MB): peak = 6312.160 ; gain = 0.996 ; free physical = 3229 ; free virtual = 18165
Restored from archive | CPU: 0.020000 secs | Memory: 0.263512 MB |
Finished XDEF File Restore: Time (s): cpu = 00:00:00.02 ; elapsed = 00:00:00.03 . Memory (MB): peak = 6312.160 ; gain = 0.996 ; free physical = 3229 ; free virtual = 18165
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 12 instances were transformed.
  RAM32M => RAM32M (inverted pins: WCLK) (RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMS32, RAMS32): 12 instances

open_run: Time (s): cpu = 00:00:17 ; elapsed = 00:00:09 . Memory (MB): peak = 6488.977 ; gain = 479.375 ; free physical = 3076 ; free virtual = 18013
open_run synth_1 -name synth_1
Design is defaulting to impl run constrset: constrs_1
Design is defaulting to synth run part: xc7v585tffg1157-1
INFO: [Netlist 29-17] Analyzing 61 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-479] Netlist was created with Vivado 2015.2
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 12 instances were transformed.
  RAM32M => RAM32M (inverted pins: WCLK) (RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMS32, RAMS32): 12 instances
```

Synthesis Schematic for the Register File

2. Data Memory Module



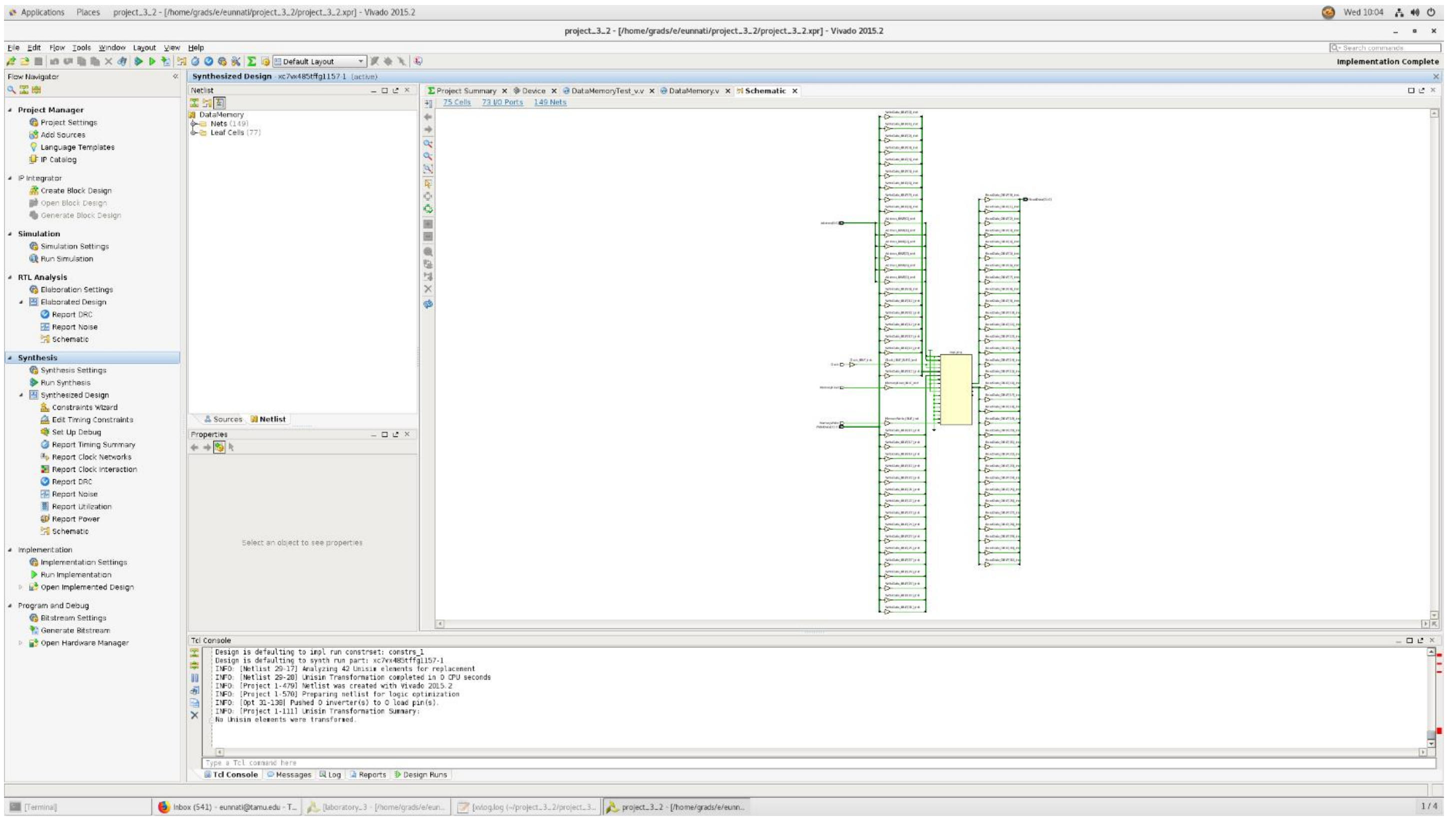
The Data Memory for the MIPS processor



Design Runs

```
Name          Constraints  Status                 Progress  WNS  TNS  WHS  THS  TPWS  Failed Routes  LUT %  LUTs  FF %   FFs  BRAM %  BRAMs  DSP %  DSPs  Start            Elapsed   Strategy                                                  Part
synth_1       constrs_1    synth_design Complete!  100%                                              0.000  0     0.000  0    0.049   0      0.000  0     9/18/19 9:52 AM  00:00:16  Vivado Synthesis Defaults (Vivado Synthesis 2015)         xc7vx485tffg1
  impl_1      constrs_1    route_design Complete!  100%     NA   NA   NA   NA   NA    0              0.000  0     0.000  0    0.049   0      0.000  0     9/18/19 9:53 AM  00:01:33  Vivado Implementation Defaults (Vivado Implementation 2015) xc7vx485tffg1
```

Schematic of Data Memory following Behavioural Verilog



Tcl Console

```
Design is defaulting to impl run constrset: constrs_1
Design is defaulting to synth run part: xc7vx485tffg1157-1
INFO: [Netlist 29-17] Analyzing 42 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-479] Netlist was created with Vivado 2015.2
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.
```

Synthesis Schematic of Data Memory following Behavioural Verilog

QUESTIONS

(a) **Suppose we could make the data memory module dual-ported (i.e. two separately addressed read ports). Could one design a microarchitecture which uses the dual-ported memory module in order to eliminate the register file? If so, would this be a good design? Explain your answer.**

A data memory module includes only one address port as the storage element because of which data cannot be written while being read and only one data element can be retrieved at a time. It contains only single read and write buses which are each 32-bits wide. If we make the data memory module dual ported i.e. having two separately addressed read ports like a register file has and use it to eliminate the register file, such a designed microarchitecture would not be performance efficient and will fail to be a good design.

Register file is responsible for holding the data that the CPU is currently processing whereas, the memory holds the program instruction and data that the program requires for execution. This register file makes it possible to simultaneously read from two registers and write into one register as it is appropriate for MIPS processor. Also in MIPS processor, the register file's "write data" input has to store either the ALU output of R-type instructions or the data memory output for load operation and hence requires a MemReg write enable control signal to select between saving the ALU result (0) or the data memory output (1) to the registers So if we replace the register file in the architecture with dual ported memory module, it will conflict with designed architecture and there will be no way to handle different write outputs and resulting in synchronisation conflicts.

Even though data memory provides the microprocessor with means of storing more long-term data, much larger in size as compared to the register file, some part of the data being processed at the register file level helps reduce latency delays. Also, the processing unit can operate on register contents at the rate of more than one operation in one clock cycle whereas memory access is slower comparatively. Hence eliminating register file might affect the performance time of the new architecture resulting in slower response time. This might need the usage of more storage elements in the form of cache or registers in the new microarchitecture to compensate for the delay.

A CPU register can generally be accessed in a single clock cycle, whereas main memory may require dozens of CPU clock cycles to read or write. Since there are very few registers compared to memory cells, registers also require far fewer bits to specify which register to use. This in turn allows for smaller instruction codes. For example, the MIPS processor has 32 general-purpose registers, so it takes 5 bits to specify which one to use. In contrast, the MIPS has a 4 gibibyte memory capacity, so it takes 32 bits to specify which memory cell to use. An instruction with 3 operands will require 15 bits if they are all registers, and 96 bits if they are all memory addresses.

Also, a load-store architecture like MIPS, only load and store instructions can access memory. All other instructions (add, sub, mul, div, and, or, etc.) must get their operands from registers and store their results in the register file. Thus, eliminating register file will not serve the purpose of arithmetic operations.

(b) What is the purpose of the data memory's MemRead, (i.e. the read enable signal)? Is it functionally necessary? What advantages might it provide? Why do we not implement a similar signal for the register file?

Memory access from the data memory module requires a single clock cycle and has synchronous read and synchronous write capabilities which includes both a read and write enable signal. MemRead signal is asserted for load instructions and tells the memory to do a read from the address port i.e it enables a memory read for load instructions. When MemRead control signal is set to high, lw: is the only instruction that reads from main memory and the MemWrite control signal must be set to low when reading operation takes place. Since data memory follows a single-cycle implementation of the processor, a single instruction requires that a register be read from and written into in the same clock period. In order to accomplish this, data memory uses edge triggered elements directly (positive edge for MemRead and negative element for MemWrite). Since all implemented instructions complete in exactly one cycle for data memory module it is functionally necessary to have MemRead as a control signal for read operation as it utilizes positive edge of clock period to execute read operation. The advantage provided is that, since a synchronous read takes place i.e its value can be changed only on a single event trigger (a positive clock edge), glitches on a control line do not lead to unexpected changes because when they take effect (on the trigger event)- the glitch will have gone away. Also, it will prevent unnecessary read operations that might take place thus increasing the access time.

A register file uses synchronous write and asynchronous read to load and store data. Register files are typically architected for concurrent read and writes and since the reading of data is not dependent upon if the writing is taking place or not, it allows for more data ports independent of clock cycle thus allowing much more parallel access with the presence of defined buses for both read input ports A & B. Also, the register file's "write data" input has to store either the ALU output of R-type instructions or the data memory output for load operation and hence requires a MemReg write enable control signal to select between saving the ALU result (0) or the data memory output (1) to the registers. This is not required for read operation and hence it doesn't require synchronous read in register file for MIPS architecture.

(c) Elaborate on the term synthesizable. What sort of constructs in Verilog are not synthesizable?

Synthesis tool in Verilog translates our written code into gates, registers, RAMs, etc. and turns it into something that the FPGA can understand and implement. Such a code is said to be synthesizable. For something to be synthesizable it has to be able to be represented in hardware, i.e. using logic gates. Only a subset of Verilog constructs can be synthesized and the code containing only this subset is synthesizable. Synthesizable Verilog includes Verilog basics, primitive cells, operators etc.

However, there are some parts of Verilog and VHDL that the FPGA simply cannot implement. When we write code like this, it is called *non-synthesizable code*. An example of something that is non-synthesizable would be initializing a design with values assigned to signals or registers. This cannot be translated to hardware, therefor is non-synthesizable. The most fundamental non-synthesizable piece of code is a delay statement. The FPGA has no concept of time, so it is impossible to tell the FPGA to wait for 5 nanoseconds**.** Instead, we need to use clocks and flip-flops to implement the functionality. Another non-synthesizable piece of code includes looping statements, such as while, for, repeat, etc. Some more types of non-synthesizable constructs include system tasks, real constants, Delay on built-in gates, Event trigger (->), delay and wait (#), initial, fork, join, force, release etc.