ECEN 651

LAB NO. 3

Storage Elements in Verilog

TA: Mr. Ye Wang

DATE: 09/18/2019

Student Name: Dhiraj Dinesh Kudva

UIN: 829009538

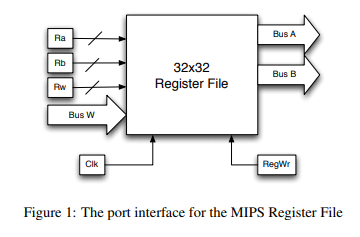
**Objective:** The objectives of this lab are

1. To learn about different types of storage elements and implement a MIPS register file and a simple data memory module.
2. Implementation of MIPS register file using Verilog: This lab will focus on implementing the register file using Verilog and synthesizing the code to check its validity.
3. Implementation of Simple Data Memory Module using Verilog: This lab will also focus on implementation of Simple Data Memory Module using Verilog and synthesizing the code to check its validity.

**Design:**

1. **32 by 32 bit register file**

The MIPS register file to be implemented is 32 by 32 bit register file. It has three buses: BusA and Bus B for output and Bus W for input. Ra, Rb and Rc store the addresses of the register. RegWr is an enable signal for writing data into register file.



**STEPS- 32 by 32 bit register file:**

1. Open the Xilinx Vivado software.
2. Create new project and in that create new source files in that project.
3. Create a separate module for 32 by 32 bit register file.
4. Define the inputs and outputs.
5. Create source code for 32 by 32 bit register file. The write command will work only when the RegWr signal is high.
6. After this step, write test bench for the same.
7. Mention the delay and clock cycle in this test bench. Initialization of the variables should be done in the test bench.
8. Run the testbench and verify the output.
9. Synthesize the source code

**Verilog Code: 32 BY 32 bit register**

`timescale 1ns / 1ps

//initializing module for Register File

module RegisterFile(BusA, BusB, BusW, RA, RB, RW, RegWr, Clk);

// declaring inputs and outputs

output [31:0] BusA, BusB;

//BusA, BusB are read buses and hence declared as outputs.

//They carry data from the register and displays the value.

input [31:0] BusW; /\* BusW will write the data into the register location specified by RW and hence considered as inputs\*/

input [4:0] RA, RB, RW; // RA,RB and RW stores address

input RegWr; //enable command for writing in register.

input Clk; //clock

reg [31:0] registers [31:0]; //32 registers each capable of storing 32 bits

initial begin

registers[0]=5'd0; //R0 is declared as zero register

end

assign #2 BusA = registers[RA]; //the value at RA location in the register is assigned to BusA.

assign #2 BusB = registers[RB]; //the value at RB location in the register is assigned to BusB.

always @ (negedge Clk) //module is triggered at negative edge

begin

if(RegWr) //condition for writing: only when RegWr is high.

begin

if (RW != 5'd0) //assigning R0 equals zero

registers[RW] <= #3 BusW; // writing the data into memory.

end

end

endmodule

**Testbench for 32 by 32 bit register (to be commented)**

`timescale 1ns / 1ps

`define STRLEN 32

module RegisterFileTest\_v;

//checking if all the tests are passed or not

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"); //when all tests are passed

else $display("Some tests failed"); //when some tests are 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)

RegisterFile 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);

//writing into the registers

{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};

//reading data from registers and checking all the conditions

#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;

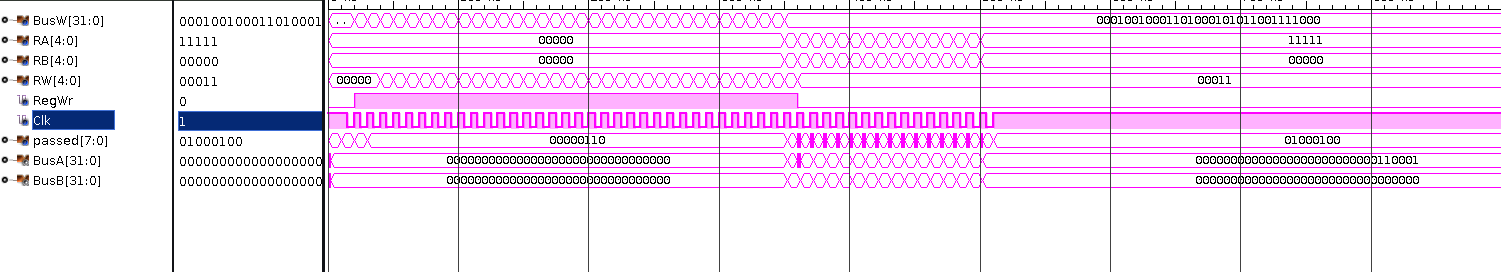
passTest(BusA, 32'h31, "Value Not Updated 16", passed);

allPassed(passed, 68);

end

endmodule

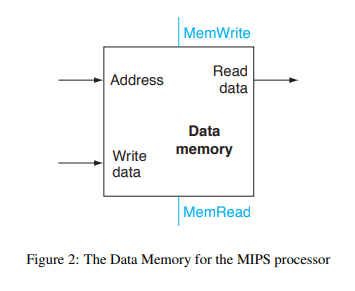
**Output 32 by 32 bit register file:**





1. **Data Memory System**

The Data Memory system provide synchronous read and synchronous write capabilities and include both a read and write enable signal. Access to the memory requires only a single clock but the read and write signals should occur at alternative clock edges.



**Steps:**

1. Open the Xilinx Vivado software.
2. Create new project and in that create new source files in that project.
3. Create a separate module for data memory system
4. Define the inputs and outputs.
5. Create source code for memory system capable of storing 256 bytyes. The write command will work only when the memwrite command is high and memread command is low and vice versa for read command.
6. The read and write command should not occur at the same edge of the clock cycle.
7. After this step, write test bench for the same.
8. Mention the delay and clock cycle in this test bench. Initialization of the variables should be done in the test bench.
9. Run the testbench and verify the output.
10. Synthesize the source code

**Verilog Code for Data Memory System**

`timescale 1ns/ 1ps

`timescale 1ns / 1ps

module data\_memory (

input wire [5:0] addr, // Memory Address

input wire [31:0] write\_data, // Memory Address Contents

input wire memwrite, memread,

input wire clk, // All synchronous elements, including memories, should have a clock signal

output reg [31:0] read\_data // Output of Memory Address Contents

);

reg [31:0] MEMO[0:63]; // 63 registers of 32 bit each

integer i;

//storing dummy vakues in data memory

initial begin

read\_data <= 0;

for (i = 0; i < 256; i = i + 1) begin

MEMO[i] = i;

end

end

// Data is written into the memory during negative edge

always @(negedge clk) begin

if (memwrite == 1'b1 && memread==1'b0) begin

//when write signal is 1 and read signal is 0

MEMO[addr] <= write\_data;

end

end

//data is read from the memory during the positive edge

always @(posedge clk) begin

if (memread == 1'b1 && memwrite==1'b0) begin

//when read signal is 1 and write signal is 0

read\_data <= MEMO[addr];

end

end

endmodule

**Testbench for Dflipflop**

`timescale 1ns / 1ps

`define STRLEN 32

module DataMemoryTest\_v;

//checking if all the tests are passed or not

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"); //when all tests are passed

else $display("Some tests failed"); // when some tests are 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)

data\_memory uut (

.read\_data(ReadData),

.addr(MemAddress),

.write\_data(WriteData),

.memread(MemoryRead),

.memwrite(MemoryWrite),

.clk(Clock)

);

initial begin

// Initialize Inputs

Address = 0;

WriteData = 0;

MemoryRead = 0;

MemoryWrite = 0;

Clock = 0;

passed = 0;

// Add stimulus here

//writing data into memory

$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 = 1;

//reading data from memory and checking conditions

{Address, WriteData, MemoryWrite, MemoryRead} = {32'h14, 32'hffff0000, 2'h1};

#50 Clock = 0;

#50 Clock = 1;

passTest(ReadData, 32'h0, "Read address 0x14", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hf0, 32'hffff0000, 2'h1};

#50 Clock = 0;

#50 Clock = 1;

passTest(ReadData, 32'hffff0000, "Read address 0xf0", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hcc, 32'hffff0000, 2'h1};

#50 Clock = 0;

#50 Clock = 1;

passTest(ReadData, 32'd1431699200, "Read address 0xcc", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hc8, 32'hffff0000, 2'h1};

#50 Clock = 0;

#50 Clock = 1;

passTest(ReadData, 32'haaaaffff, "Read address 0xc8", passed);

{Address, WriteData, MemoryWrite, MemoryRead} = {32'hc, 32'hffff0000, 2'h1};

#50 Clock = 0;

#50 Clock = 1;

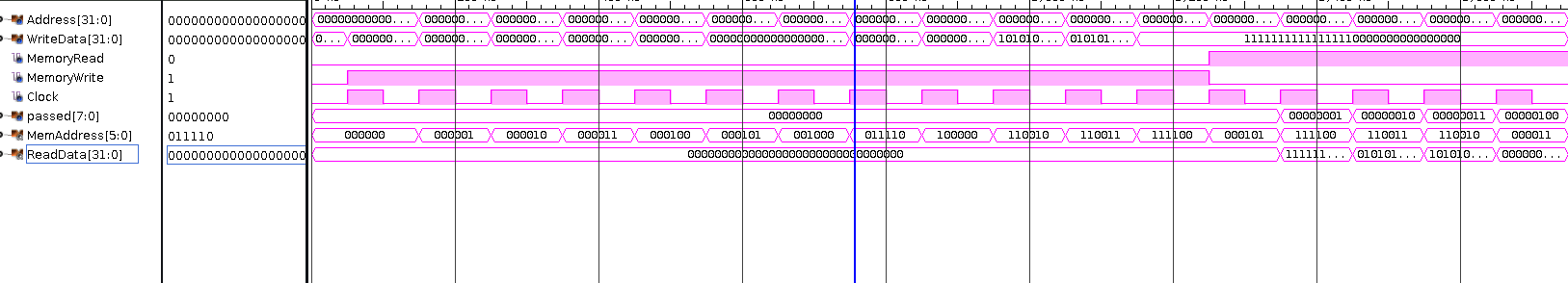
passTest(ReadData, 32'd40, "Read address 0xc", passed);

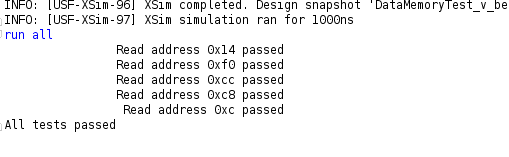
allPassed(passed, 5);

end

endmodule

**Output for Dflipflop**





**Questions:**

1. 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.

Ans. The data memory module with dual ported can be integrated with the microarchitecture. But the time to fetch the data from memory and again write it back would be greater as compared to the microarchitecture with register. The data memory are also large in size compared to registers. So it would be difficult to fabricate a large data memory along with microprocessor on a single chip.

1. 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?

Ans: The MemRead allows to read the data from the memory only when it is enabled. The MemRead also prevents case where both read and write are working on the same data. If there was no MemRead, then there might be a possibility that both read and write are operating at the same data and this might cause difference in the value. Such difference will provide unwanted output. The read command in register file is given by the processor whereas in data memory it is given by the user. So, it is not needed in the register file as there is very less chance of getting any unwanted output. Hence for this reason, MemRead was used in data memory.

1. Elaborate on the term synthesizable. What sort of constructs in Verilog are not synthesizable?

Ans: The synthesizable in Verilog refers to those constructs that can be represented in hardware components like gates and registers. If the constructs can not be represented in hardware, then they are called as ‘non synthesizable’ constructs. The non synthesizable constructs in Verilog are initial, time, delays, force and release, and real.