**Jared Hayes**

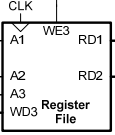
**CPE 408: VLSI Physical Design, Verification and Testing**

**Midterm 1: 8x16 Register File**

March 18, 2015

**Code\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

My device under verification is a 3-port register file that can read from two registers and write to one register at once. It is a 16-bit register file with eight registers in total. To avoid data mismatch, write enable is only asserted on the positive edge of the clock.



*DUT.v file*

`timescale 1ns / 1ps

module regfile

(input clk,

input write, // Write enable

input [2:0] wrAddr, // Address of register to be written to

input [15:0] wrData, // Data to be written

input [2:0] rdAddrA, // Address of Port A register

output [15:0] rdDataA, // Data in Port A register

input [2:0] rdAddrB, // Address of Port B register

output [15:0] rdDataB); // Data in Port B register

reg [15:0] regfile [0:7]; // 8 by 16 register file

assign rdDataA = regfile[rdAddrA]; // Read Port A output from register address 1

assign rdDataB = regfile[rdAddrB]; // Read Port A output from register address 2

always @(posedge clk) // Write enable only on positive edge of clk

begin

if (write)

regfile[wrAddr] <= wrData; // Write to register

end

endmodule

*DUT\_tb.v file*

`timescale 1ns / 1ps

`define EOF 32'hFFFF\_FFFF

`define NEWLINE 10

`define NULL 0

`ifndef INPUT

`define INPUT "test\_regfile.input"

`endif

`define OUTPUT "test\_regfile.output"

// Testbench for 8 by 16 register file

module testbench\_v;

integer input\_file, output\_file, errors, linenum;

// Inputs

reg write;

reg clk;

reg [2:0] rdAddrA;

reg [2:0] rdAddrB;

reg [2:0] wrAddr;

reg [15:0] wrData;

// Outputs

wire [15:0] rdDataA;

wire [15:0] rdDataB;

// Instantiate the Device Under Verification (DUV)

regfile DUV (.clk(clk),

.write(write),

.wrAddr(wrAddr),

.wrData(wrData),

.rdAddrA(rdAddrA),

.rdDataA(rdDataA),

.rdAddrB(rdAddrB),

.rdDataB(rdDataB));

reg [15:0] expectedValue1;

reg [15:0] expectedValue2;

always #5 clk <= ~clk; // Instantiate clock

initial begin

// Initialize Inputs

clk = 0;

write = 0;

wrAddr = 0;

rdAddrA = 0;

rdAddrB = 0;

wrData = 0;

errors = 0;

linenum = 0;

output\_file = 0;

// open the test inputs

input\_file = $fopen(`INPUT, "r");

if (input\_file == `NULL) begin

$display("Error opening file: ", `INPUT);

$finish;

end

// open the output file

`ifdef OUTPUT

output\_file = $fopen(`OUTPUT, "w");

if (output\_file == `NULL) begin

$display("Error opening file: ", `OUTPUT);

$finish;

end

`endif

// Check test cases

#100;

while (7 == $fscanf(input\_file, "%d %d %d %b %h %h %h", rdAddrA, rdAddrB, wrAddr, write, wrData, expectedValue1, expectedValue2))

begin // Read test cases

#8;

linenum = linenum + 1; // Track line numbers

if (output\_file) begin // Output test results

$fdisplay(output\_file, "%d %d %d %b %h %h %h", rdAddrA, rdAddrB, wrAddr, write, wrData, rdDataA, rdDataB);

end

if (rdDataA != expectedValue1) // Check if Readdata on Port A matches expected value

begin

$display("Error at line %d: Value of register %d on port A should have been %d, but was %d instead", linenum, rdAddrA, expectedValue1, rdDataA);

errors = errors + 1; // Display error if data mismatch and track errors

end

if (rdDataB != expectedValue2) begin // Check if Readdata on Port B matches expected value

$display("Error at line %d: Value of register %d on port A should have been %d, but was %d instead", linenum, rdAddrB, expectedValue2, rdDataB);

errors = errors + 1; // Display error if data mismatch and track errors

end

#2;

end // end while

if (input\_file) $fclose(input\_file); // Close input file

if (output\_file) $fclose(output\_file); // Close output file

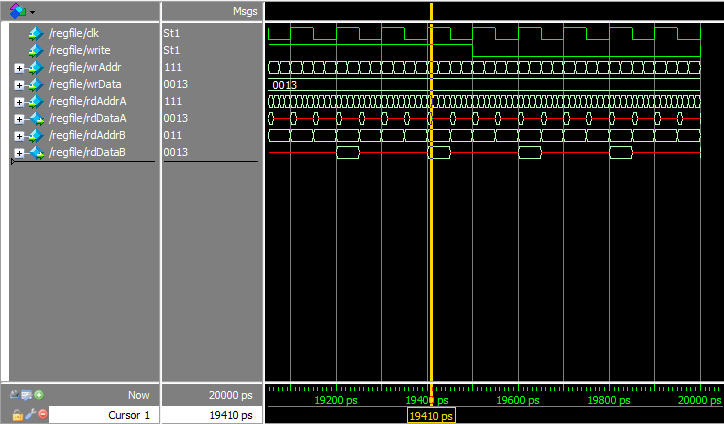
$display("Simulation finished: %d test cases, %d errors [%s]", linenum, errors, `INPUT); // Display results

$finish;

end

endmodule

Example Simulation



Example Test Vectors (found in test\_regfile.input file)

**Format:** rdAddrA, rdAddrB, wrAddr, write(Enable) wrData, expectvalA, expectvalB (hex)

0 1 7 1 003f 0001 0002

5 0 2 0 bc8a 0006 0001

5 7 0 0 07a0 0006 003f

Results

There are 512 permutations of register accesses and 1024 different cases when taking the “Write Enable” signal into account. The input file specified in the testbench generated 1011 of these cases in no particular order and tests all of the unique cases. My terminal does not display any of the errors that my testbench is programmed to generate due to data mismatches. The data read from the two output ports matches expected values for all test vectors in the input file.