Lab 3: Register File

T 2pm-5pm

Zichen Huang and Jack Landers

Introduction

In this lab we designed a register file with 8 8-bit registers for writing and reading, with a bypass function when both were enabled for the same register.

Procedure

For this design we created a module for the 8-bit register which would output the bits when load enable is true. To apply 8 of these, 3 bit address values were used in case statements and the read enable and write enable values were one hot encoded for each register. This would be such that if we wanted to read or write an address, only the bit of that respective read or write enable would be set and used for the register. For the case of reading and writing enabled for a register, a bypass was achieved by checking that these enables were equal at the same time as load enable being true. This case works because previously when writing and reading we do not need to check the case of load enable being true.

Results

We tested these operations work by writing values to all of our registers and reading them. We then tested the case of bypass, where we could see the value writing and reading but not changing on the next read operation. To confirm the registers are not hard coded to those values we also retest by writing new values to them.

What problems did you encounter while testing your steps yourself?

There was an issue running the bypass where we continued to check if the load enable value was true when checking read and write cases, causing them to not always work

Did any problems arise when demonstrating for the TA? What were they? Explain your thoughts on how/why these test cases escaped your own testing.

We extended our test to show that the registers were able to change after being written once.

Final source code for register file:

```
module register_file (write_data, write_addr, load_enable, read_addr0, read_addr1,
read_data0, read_data1, CLK);

input  [7:0] write_data;
input  [2:0] write_addr;
input  load_enable;
input  [2:0] read_addr0;
input  [2:0] read_addr1;
input  CLK;

wire [7:0] read_data_bus1;
wire [7:0] read_data_bus2;
wire [7:0] read_data_bus3;
wire [7:0] read_data_bus4;
wire [7:0] read_data_bus5;
wire [7:0] read_data_bus6;
wire [7:0] read_data_bus7;
wire [7:0] read_data_bus8;
reg [7:0] load_enable_bus;

output reg[7:0] read_data0;
output reg [7:0] read_data1;

reg8 r1(load_enable_bus[0], CLK, write_data, read_data_bus1);
reg8 r2(load_enable_bus[1], CLK, write_data, read_data_bus2);
reg8 r3(load_enable_bus[2], CLK, write_data, read_data_bus3);
reg8 r4(load_enable_bus[3], CLK, write_data, read_data_bus4);
reg8 r5(load_enable_bus[4], CLK, write_data, read_data_bus5);
reg8 r6(load_enable_bus[5], CLK, write_data, read_data_bus6);
reg8 r7(load_enable_bus[6], CLK, write_data, read_data_bus7);
reg8 r8(load_enable_bus[7], CLK, write_data, read_data_bus8);

always @(posedge CLK) begin
        if(load_enable) begin
                case(write_addr)
                        3'b000: load_enable_bus = 8'b00000001;
                        3'b001: load_enable_bus = 8'b00000010;
                        3'b010: load_enable_bus = 8'b00000100;
                        3'b011: load_enable_bus = 8'b00001000;
```

```
				3'b100:		load_enable_bus = 8'b00010000;
				3'b101: load_enable_bus = 8'b00100000;
				3'b110:		load_enable_bus = 8'b01000000;
				3'b111: load_enable_bus = 8'b10000000;
			endcase
		end else begin
			load_enable_bus = 8'b00000000;
		end

		if(load_enable && read_addr0 == write_addr) begin
			read_data0 = write_data;
		end else begin
			case(read_addr0)
				3'b000: read_data0 = read_data_bus1;
				3'b001: read_data0 = read_data_bus2;
				3'b010: read_data0 = read_data_bus3;
				3'b011: read_data0 = read_data_bus4;
				3'b100: read_data0 = read_data_bus5;
				3'b101: read_data0 = read_data_bus6;
				3'b110: read_data0 = read_data_bus7;
				3'b111: read_data0 = read_data_bus8;
			endcase
		end

		if (load_enable && read_addr1 == write_addr) begin
			read_data1 = write_data;
		end else begin
			case(read_addr1)
				3'b000: read_data1 = read_data_bus1;
				3'b001: read_data1 = read_data_bus2;
				3'b010: read_data1 = read_data_bus3;
				3'b011: read_data1 = read_data_bus4;
				3'b100: read_data1 = read_data_bus5;
				3'b101: read_data1 = read_data_bus6;
				3'b110: read_data1 = read_data_bus7;
				3'b111: read_data1 = read_data_bus8;
			endcase
		end
	end
```

```
endmodule
```

Final source code for the testbench:

```
module reg_tb();

reg  [7:0] write_data;
reg  [2:0] write_addr;
reg  load_enable;
reg  [2:0] read_addr0;
reg  [2:0] read_addr1;
reg  CLK;

wire [7:0] read_data0;
wire [7:0] read_data1;

register_file reg1(write_data, write_addr, load_enable, read_addr0, read_addr1,
read_data0, read_data1, CLK);

always begin
        #5 CLK <= !CLK;
end

initial begin
$display("Start of the bcd adder test");
$monitor("load_enable=%b, CLK=%b, write_data=%h, write_addr=%d,
read_addr0=%d, read_addr1=%d, read_data0=%h, read_data1=%h, @ %d",
load_enable, CLK, write_data, write_addr, read_addr0, read_addr1 ,read_data0,
read_data1, $time);
CLK = 0;
load_enable = 1; write_data = 8'h10; write_addr = 3'b000; read_addr0 = 3'b000;
read_addr1 = 3'b000;
#10
load_enable = 1; write_data = 8'h20; write_addr = 3'b001; read_addr0 = 3'b000;
read_addr1 = 3'b000;
#10
load_enable = 1; write_data = 8'h30; write_addr = 3'b010; read_addr0 = 3'b001;
read_addr1 = 3'b001;
#10
```

```
load_enable = 1; write_data = 8'h40; write_addr = 3'b011; read_addr0 = 3'b010;
read_addr1 = 3'b010;
#10
load_enable = 1; write_data = 8'h50; write_addr = 3'b100; read_addr0 = 3'b011;
read_addr1 = 3'b011;
#10
load_enable = 1; write_data = 8'h60; write_addr = 3'b101; read_addr0 = 3'b100;
read_addr1 = 3'b100;
#10
load_enable = 1; write_data = 8'h70; write_addr = 3'b110; read_addr0 = 3'b101;
read_addr1 = 3'b101;
#10
load_enable = 1; write_data = 8'h80; write_addr = 3'b111; read_addr0 = 3'b110;
read_addr1 = 3'b110;
#10
load_enable = 0; write_data = 8'h80; write_addr = 3'b111; read_addr0 = 3'b111;
read_addr1 = 3'b111;
#10
load_enable = 1; write_data = 8'h11; write_addr = 3'b001; read_addr0 = 3'b001;
read_addr1 = 3'b000;
#10
load_enable = 1; write_data = 8'h22; write_addr = 3'b010; read_addr0 = 3'b000;
read_addr1 = 3'b010;
#10
load_enable = 1; write_data = 8'h33; write_addr = 3'b011; read_addr0 = 3'b011;
read_addr1 = 3'b011;
#10
load_enable = 0; write_data = 8'h38; write_addr = 3'b100; read_addr0 = 3'b011;
read_addr1 = 3'b011;
#10
load_enable = 0; write_data = 8'h33; write_addr = 3'b011; read_addr0 = 3'b100;
read_addr1 = 3'b100;
#10
load_enable = 1; write_data = 8'h15; write_addr = 3'b000; read_addr0 = 3'b000;
read_addr1 = 3'b000;
#10
load_enable = 1; write_data = 8'h25; write_addr = 3'b001; read_addr0 = 3'b000;
read_addr1 = 3'b000;
#10
```

```
load_enable = 1; write_data = 8'h35; write_addr = 3'b010; read_addr0 = 3'b001;
read_addr1 = 3'b001;
#10
load_enable = 1; write_data = 8'h45; write_addr = 3'b011; read_addr0 = 3'b010;
read_addr1 = 3'b010;
#10
load_enable = 1; write_data = 8'h55; write_addr = 3'b100; read_addr0 = 3'b011;
read_addr1 = 3'b011;
#10
load_enable = 1; write_data = 8'h65; write_addr = 3'b101; read_addr0 = 3'b100;
read_addr1 = 3'b100;
#10
load_enable = 1; write_data = 8'h75; write_addr = 3'b110; read_addr0 = 3'b101;
read_addr1 = 3'b101;
#10
load_enable = 1; write_data = 8'h85; write_addr = 3'b111; read_addr0 = 3'b110;
read_addr1 = 3'b110;
#10
load_enable = 0; write_data = 8'h85; write_addr = 3'b111; read_addr0 = 3'b111;
read_addr1 = 3'b111;
#10
$finish;
end
endmodule
```