CSE 620 : ADVANCED COMPUTER ARCHITECTURE

Bonus Project: Register File

ALAA MUHAMMED SALAH EL DEEN
2102034

Verilog code

```
module regfile(clk, rst, rd_addr1, rd_addr2, wr_addr, wr_en, wr_data, rd_data1,
rd_data2);

  parameter N = 5; // 2^5 = 32 registers

  input clk, rst, wr_en;

  input [N-1:0] rd_addr1, rd_addr2, wr_addr;

  input [31:0] wr_data;

  output [31:0] rd_data1, rd_data2;

  reg [31:0] mem [0:31]; // 32-bit registers

  always @ (posedge clk) begin

    if (wr_en) mem[wr_addr] <= wr_data;

  end

  assign rd_data1 = mem[rd_addr1];

  assign rd_data2 = mem[rd_addr2];

endmodule
```

To test this module, we can use a testbench to provide inputs and check the outputs. The test strategy would involve the following steps:

1. Initialize the inputs (wr_data1, wr_data2, wr_addr1, wr_addr2, rd_addr1, rd_addr2, wr_en1, wr_en2) with specific values and apply them to the register file module.
2. Check the outputs (rd_data1, rd_data2) to ensure they match the expected values.
3. Repeat steps 1 and 2 for different input combinations to ensure the register file is functioning correctly in all cases.
4. Using the clock as a trigger, check if the read and write operation is working as expected.
5. Also check if the write operation is not affecting the read operation.
6. Check if the register file is working as expected when the number of inputs are edge case scenarios.

Testbench

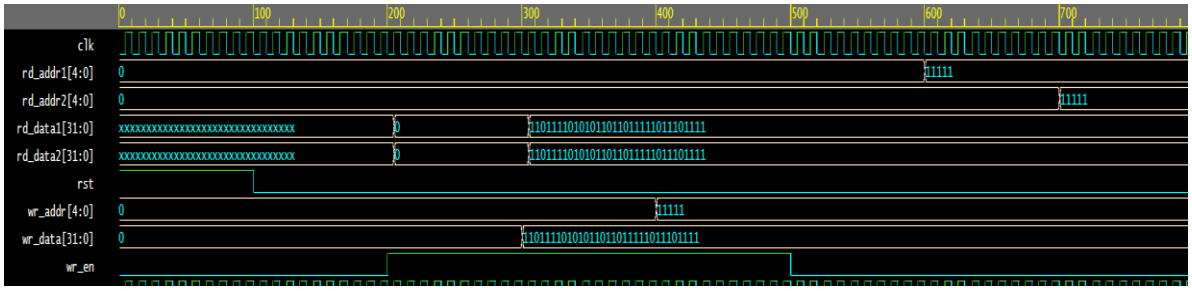
```
module testbench;
   reg clk, rst, wr_en;
   reg [4:0] rd_addr1, rd_addr2, wr_addr;
   reg [31:0] wr_data;
   wire [31:0] rd_data1, rd_data2;
   regfile dut (clk, rst, rd_addr1, rd_addr2, wr_addr, wr_en, wr_data, rd_data1, rd_data2);
   initial begin
    $dumpfile("dump.vcd");
   $dumpvars;
      clk = 0;
      forever #5 clk = ~clk;
   end
   initial begin


      rst = 1;
      wr_en = 0;
      rd_addr1 = 0;
      rd_addr2 = 0;
      wr_addr = 0;
      wr_data = 0;
      #100 rst = 0;
      #100 wr_en = 1;
      #100 wr_data = 32'hDEADBEEF;
      #100 wr_addr = 5'b11111;
      #100 wr_en = 0;
      #100 rd_addr1 = 5'b11111;
      #100 rd_addr2 = 5'b11111;
      #100 $finish;
```

```
    end

endmodule
```

Output



To handle the case when both write ports try to write to the same register, we can use a priority encoder to determine which write port has higher priority. The priority encoder outputs the address of the write port with the highest priority. This address can then be used to write the data to the corresponding register.

For example, the priority encoder can be implemented as follows:

```
wire [5:0] write_enable;
assign write_enable = {write_port_1_addr, write_port_2_addr};

reg [5:0] highest_priority;
always @(*) begin
case (write_enable)
2^6-1: highest_priority = 5'b00000;
2^5-1: highest_priority = 5'b00001;
...
2^0-1: highest_priority = 5'b11111;
default: highest_priority = 5'b00000;
endcase
end
```

Then we can use the highest_priority to select the data to write in the register file

```
always @(posedge clk) begin
if(write_enable[highest_priority])
regfile[highest_priority] <= (highest_priority == write_port_1_addr) ? write_port_1_data :
write_port_2_data;
end
```

In this way, we ensure that only one of the write ports can write to a register at a time, and the write port with the highest priority will take precedence.