Course: CS-223 Digital Design
Section 01
Lab 05

Miray Ayerdem
21901987
14/04/2021

a. Write the Systemverilog code for RegisterFile, ALU and MUX modules (Basically, all modules).

```
module registerfile( input logic clk, wr_en, reset, [2:0] rda1, rda2, wra, [3:0] wrd, output logic [3:0] rdd1 ,rdd2);
   logic [7:0] addr;
   logic [3:0] reg_dt0, reg_dt1, reg_dt2, reg_dt3, reg_dt4, reg_dt5, reg_dt6, reg_dt7;
   decoder3_8 address_for_w ( wra[2], wra[1], wra[0], wr_en, addr );

   reg4bit address0( clk, addr[0], reset, wrd, reg_dt0 );
   reg4bit address1( clk, addr[1], reset, wrd, reg_dt1 );
   reg4bit address2( clk, addr[2], reset, wrd, reg_dt2 );
   reg4bit address3( clk, addr[3], reset, wrd, reg_dt3 );
   reg4bit address4( clk, addr[4], reset, wrd, reg_dt4 );
   reg4bit address5( clk, addr[5], reset, wrd, reg_dt5 );
   reg4bit address6( clk, addr[6], reset, wrd, reg_dt6 );
   reg4bit address7( clk, addr[7], reset, wrd, reg_dt7 );

   mux8_1 reading_1( reg_dt0,  reg_dt1, reg_dt2,  reg_dt3,  reg_dt4,  reg_dt5 , reg_dt6, reg_dt7, rda1, rdd1 );
   mux8_1 reading_2( reg_dt0,  reg_dt1,  reg_dt2,  reg_dt3,  reg_dt4,  reg_dt5 , reg_dt6, reg_dt7, rda2, rdd2 );

endmodule



module reg4bit(input logic[3:0] d, input logic clk, load, reset, output logic [3:0] q);
   always@(posedge reset or posedge clk)
     if (reset)
       begin
       q[0] <=  0;
       q[1] <=  0;
       q[2] <=  0;
       q[3] <=  0;
       end
     else
        begin
        if (load)
        begin
        q[3] <=  d[3];
        q[2] <=  d[2];
        q[1] <=  d[1];
        q[0] <=  d[0];
        end
     end
endmodule
```

```
module mux8_1 (input logic [3:0] d0, d1, d2, d3, d4, d5, d6, d7, [2:0] s, output logic [3:0] y);
   assign y = s[2] ? ( s[1] ? ( s[0] ? d7 : d6 ) : ( s[0] ? d5 : d4 ) ) : ( s[1] ? ( s[0] ? d3 : d2 ) : (
s[0] ? d1 : d0 ) );
endmodule



module ALU( input logic [3:0] a, b, input logic [1:0] sel, output logic [3:0] rel
   );

   logic[3:0] cout0, cout1, cout2;
   logic[3:0] temp0, temp1, temp2, temp3;
   full_adder addition_a0_1(a[0],1,0, temp0[0], cout0[0]);
   full_adder addition_a1_1(a[1],0,cout0[0], temp0[1], cout0[1]);
   full_adder addition_a2_1(a[2],0, cout0[1], temp0[2], cout0[2]);
   full_adder addition_a3_1(a[3],0, cout0[2], temp0[3], cout0[3]);

   full_adder addition_a0_b0(a[0],b[0],0, temp1[0], cout1[0]);
   full_adder addition_a1_b1(a[1],b[1], cout1[0], temp1[1],cout1[1]);
   full_adder addition_a2_b2(a[2],b[2], cout1[1],temp1[2], cout1[2]);
   full_adder addition_a3_b3(a[3],b[3], cout1[2], temp1[3], cout1[3]);

   full_adder subtraction_a0_b0(a[0], ~b[0], 1, temp2[0], cout2[0]);
   full_adder subtraction_a1_b1(a[1], ~b[1], cout2[0], temp2[1], cout2[1]);
   full_adder subtraction_a2_b2(a[2], ~b[2], cout2[1], temp2[2], cout2[2]);
   full_adder subtraction_a3_b3(a[3], ~b[3], cout2[2], temp2[3], cout2[3]);

   assign temp3[0] = a[0] | b[0];
   assign temp3[1] = a[1] | b[1];
   assign temp3[2] = a[2] | b[2];
   assign temp3[3] = a[3] | b[3];

   mux4_1 mux1(temp0, temp1, temp2, temp3, sel, rel);
endmodule



module full_adder(input logic a, b, cin, output logic s, cout );
   assign s = a ^ b ^ cin;
   assign cout = (a & b) | (a & cin) | (b & cin);
endmodule



module mux4_1( input logic[3:0] d0, d1, d2, d3, input logic[1:0] s, output logic[3:0] y);
   assign y = s[1] ? ( s[0] ? d3: d2 ) : ( s[0] ? d1: d0);
```

```
endmodule
module decoder3_8( input logic a, b, c, en, output logic [7:0] out );

    assign out[0] = ~a & ~b & ~c & en;
    assign out[1] = ~a & ~b & c & en;
    assign out[2] = ~a & b & ~c & en;
    assign out[3] = ~a & b & c & en;
    assign out[4] = a & ~b & ~c & en;
    assign out[5] = a & ~b & c & en;
    assign out[6] = a & b & ~c & en;
    assign out[7] = a & b & c & en;

endmodule
```

b. Write test benches for your ALU and RegisterFile modules.

```
module tb_alu();
logic [1:0] sel;
logic [3:0] a, b, rel;

ALU test(a, b,sel,rel);
    initial begin
         sel = 2'b01; a = 4'b0011; b = 4'b0001;
        #100;
        sel = 2'b01; a = 4'b0011; b = 4'b0101;
        #100;
        sel = 2'b10; a = 4'b0011; b = 4'b0001;
        #100
        sel = 2'b00; a = 4'b0011; b = 4'b0001;
        #100;
        sel = 2'b00; a = 4'b1010; b = 4'b0001;
        #100;
        sel = 2'b10; a = 4'b0011; b = 4'b0101;
        #100;
        sel = 2'b11; a = 4'b0011; b = 4'b0101;
        #100;
        sel = 2'b11; a = 4'b0011; b = 4'b0001;
        #100;
    end
endmodule
```

```
module tb_regfile();
    logic [2:0] rda1;
    logic [2:0] rda2;
    logic [2:0] wra;
    logic [3:0] wrd;
    logic [3:0] rdd1;
    logic [3:0] rdd2;
    logic wr_en;
    logic clk;
    logic rst;


    registerfile regfiletb(clk, wr_en, rst, rda1, rda2, wra, wrd, rdd1, rdd2);

    always #10 clk = ~clk;

     initial
        begin
        clk <= 0;
        wr_en <= 1;
        rst <= 1;
        rst <= 0;
        #20;

        wra <= 3'b000;
        wrd <= 4'b0000;
        rda1 <= 3'b000;
        #20;

        wra <= 3'b001;
        wrd <= 4'b0001;
        rda2 <= 3'b001;
        #20;


        wra <= 3'b001;
        wrd <= 4'b0010;
        rda1 <= 3'b001;
        rda2 <= 3'b000;
        #20;

        wra <= 3'b011;
        wrd <= 4'b1010;
        rda1 <= 3'b011;
        rda2 <= 3'b001;
        end
endmodule
```

c. Write a top module for the datapath and an overall test bench. Instead of the 7-segment display output, show the ALU output in your simulation.

```
module tb_topmodule();

    logic[3:0] extData;
    logic isExternal;
    logic pushButn;
    logic clk;
    logic[1:0] ALUSel;
    logic reset;
    logic[3:0] ALUOutput;
    logic[2:0] AddrSrc1;
    logic[2:0] AddrSrc2;
    logic[2:0] AddrDest;


TopModule tbtopmodule(extData, AddrSrc1, AddrSrc2,  AddrDest, isExternal, clk, pushButn,
reset, ALUSel, ALUOutput);


    initial
        clk = 1;

    always
        begin
        #1;

        clk = ~clk;

        end

    initial begin
    reset = 1; #10;
    reset = 0; #10;
    extData = 1; isExternal = 1; AddrSrc1 = 1; AddrSrc2 = 1; AddrDest = 0;  pushButn = 0;
ALUSel = 2'b00; #10;
    extData = 3; isExternal = 1; AddrSrc1 = 1; AddrSrc2 = 1; AddrDest = 1;  pushButn = 1;
ALUSel = 2'b01; #10;
    extData = 2; isExternal = 1; AddrSrc1 = 1; AddrSrc2 = 1; AddrDest = 2;  pushButn = 0;
ALUSel = 2'b10; #10;
    extData = 0; isExternal = 1; AddrSrc1 = 1; AddrSrc2 = 1; AddrDest = 3;  pushButn = 1;
ALUSel = 2'b11; #10;

    end
endmodule
```