**Digital Design and Computer Organization Laboratory**

**UE20CS206**

**3rd Semester, Academic Year 2020-21**

Date:

|  |  |  |
| --- | --- | --- |
| Name : Sriram R | SRN : PES1UG20CS435 | Section : H |

Experiment Number: 1 Week # : 5

**Title of the Program:**

**Code :**

**reg\_alu.v :**

// Write code for modules you need here

module invert (input wire i, output wire o);

assign o = !i;

endmodule

module and2 (input wire i0, i1, output wire o);

assign o = i0 & i1;

endmodule

module or2 (input wire i0, i1, output wire o);

assign o = i0 | i1;

endmodule

module xor2 (input wire i0, i1, output wire o);

assign o = i0 ^ i1;

endmodule

module nand2 (input wire i0, i1, output wire o);

wire t;

and2 and2\_0 (i0, i1, t);

invert invert\_0 (t, o);

endmodule

module nor2 (input wire i0, i1, output wire o);

wire t;

or2 or2\_0 (i0, i1, t);

invert invert\_0 (t, o);

endmodule

module xnor2 (input wire i0, i1, output wire o);

wire t;

xor2 xor2\_0 (i0, i1, t);

invert invert\_0 (t, o);

endmodule

module and3 (input wire i0, i1, i2, output wire o);

wire t;

and2 and2\_0 (i0, i1, t);

and2 and2\_1 (i2, t, o);

endmodule

module or3 (input wire i0, i1, i2, output wire o);

wire t;

or2 or2\_0 (i0, i1, t);

or2 or2\_1 (i2, t, o);

endmodule

module nor3 (input wire i0, i1, i2, output wire o);

wire t;

or2 or2\_0 (i0, i1, t);

nor2 nor2\_0 (i2, t, o);

endmodule

module nand3 (input wire i0, i1, i2, output wire o);

wire t;

and2 and2\_0 (i0, i1, t);

nand2 nand2\_1 (i2, t, o);

endmodule

module xor3 (input wire i0, i1, i2, output wire o);

wire t;

xor2 xor2\_0 (i0, i1, t);

xor2 xor2\_1 (i2, t, o);

endmodule

module xnor3 (input wire i0, i1, i2, output wire o);

wire t;

xor2 xor2\_0 (i0, i1, t);

xnor2 xnor2\_0 (i2, t, o);

endmodule

module mux2 (input wire i0, i1, j, output wire o);

assign o = (j==0)?i0:i1;

endmodule

module mux4 (input wire [0:3] i, input wire j1, j0, output wire o);

wire t0, t1;

mux2 mux2\_0 (i[0], i[1], j1, t0);

mux2 mux2\_1 (i[2], i[3], j1, t1);

mux2 mux2\_2 (t0, t1, j0, o);

endmodule

module mux8 (input wire [0:7] i, input wire j2, j1, j0, output wire o);

wire t0, t1;

mux4 mux4\_0 (i[0:3], j2, j1, t0);

mux4 mux4\_1 (i[4:7], j2, j1, t1);

mux2 mux2\_0 (t0, t1, j0, o);

endmodule

module demux2 (input wire i, j, output wire o0, o1);

assign o0 = (j==0)?i:1'b0;

assign o1 = (j==1)?i:1'b0;

endmodule

module demux4 (input wire i, j1, j0, output wire [0:3] o);

wire t0, t1;

demux2 demux2\_0 (i, j1, t0, t1);

demux2 demux2\_1 (t0, j0, o[0], o[1]);

demux2 demux2\_2 (t1, j0, o[2], o[3]);

endmodule

module demux8 (input wire i, j2, j1, j0, output wire [0:7] o);

wire t0, t1;

demux2 demux2\_0 (i, j2, t0, t1);

demux4 demux4\_0 (t0, j1, j0, o[0:3]);

demux4 demux4\_1 (t1, j1, j0, o[4:7]);

endmodule

module fulladd(input wire a, b, cin, output wire sum, cout);

wire [4:0] t;

xor2 x0(a, b, t[0]);

xor2 x1(t[0], cin, sum);

and2 a0(a, b, t[1]);

and2 a1(a, cin, t[2]);

and2 a2(b, cin, t[3]);

or2 o0(t[1], t[2], t[4]);

or2 o1(t[3], t[4], cout);

endmodule

// Write code for modules you need here

module alu1(input wire [1:0] op, input wire i0, i1, cin, output wire o, cout);

// Declare wires here

wire xorOut;

wire fullAddSum;

wire andOut;

wire orOut;

wire mux1Out;

wire mux2Out;

// Instantiate modules here

xor2 x1(i1, op[0], xorOut);

fulladd f1(i0, xorOut, op[0], fullAddSum, cout);

and2 a1(i0, i1, andOut);

or2 o1(i0, i1, orOut);

mux2 m1(andOut, orOut, op[0], mux1Out);

mux2 m2(mux1Out, fullAddSum, op[1], o);

endmodule

module alu(input wire [1:0] op, input wire [15:0] i0, i1, output wire [15:0] o, output wire cout);

// Declare wires here

wire [14:0]c;

// Instantiate modules here

alu1 a1(i0[0],i1[0],op[0],op[1],o[0],c[0]);

alu1 a2(i0[1],i1[1],c[0],op[1],o[1],c[1]);

alu1 a3(i0[2],i1[2],c[1],op[1],o[2],c[2]);

alu1 a4(i0[3],i1[3],c[2],op[1],o[3],c[3]);

alu1 a5(i0[4],i1[4],c[3],op[1],o[4],c[4]);

alu1 a6(i0[5],i1[5],c[4],op[1],o[5],c[5]);

alu1 a7(i0[6],i1[6],c[5],op[1],o[6],c[6]);

alu1 a8(i0[7],i1[7],c[6],op[1],o[7],c[7]);

alu1 a9(i0[8],i1[8],c[7],op[1],o[8],c[8]);

alu1 a10(i0[9],i1[9],c[8],op[1],o[9],c[9]);

alu1 a11(i0[10],i1[10],c[9],op[1],o[10],c[10]);

alu1 a12(i0[11],i1[11],c[10],op[1],o[11],c[11]);

alu1 a13(i0[12],i1[12],c[11],op[1],o[12],c[12]);

alu1 a14(i0[13],i1[13],c[12],op[1],o[13],c[13]);

alu1 a15(i0[14],i1[14],c[13],op[1],o[14],c[14]);

alu1 a16(i0[15],i1[15],c[14],op[1],o[15],cout);

endmodule

module df (input wire clk, in, output wire out);

reg df\_out;

always@(posedge clk) df\_out <= in;

assign out = df\_out;

endmodule

module dfr (input wire clk, reset, in, output wire out);

wire reset\_, df\_in;

invert invert\_0 (reset, reset\_);

and2 and2\_0 (in, reset\_, df\_in);

df df\_0 (clk, df\_in, out);

endmodule

module dfrl (input wire clk, reset, load, in, output wire out);

wire \_in;

mux2 mux2\_0(out, in, load, \_in);

dfr dfr\_1(clk, reset, \_in, out);

endmodule

module dfrl16(input wire clk,reset,load,input wire [15:0] in ,output wire [15:0] out);

dfrl f0(clk,reset,load,in[0],out[0]);

dfrl f1(clk,reset,load,in[1],out[1]);

dfrl f2(clk,reset,load,in[2],out[2]);

dfrl f3(clk,reset,load,in[3],out[3]);

dfrl f4(clk,reset,load,in[4],out[4]);

dfrl f5(clk,reset,load,in[5],out[5]);

dfrl f6(clk,reset,load,in[6],out[6]);

dfrl f7(clk,reset,load,in[7],out[7]);

dfrl f8(clk,reset,load,in[8],out[8]);

dfrl f9(clk,reset,load,in[9],out[9]);

dfrl f10(clk,reset,load,in[10],out[10]);

dfrl f11(clk,reset,load,in[11],out[11]);

dfrl f12(clk,reset,load,in[12],out[12]);

dfrl f13(clk,reset,load,in[13],out[13]);

dfrl f14(clk,reset,load,in[14],out[14]);

dfrl f15(clk,reset,load,in[15],out[15]);

endmodule

module mux128\_16(input wire [15:0]i0,i1,i2,i3,i4,i5,i6,i7,input wire s0,s1,s2,output wire [15:0]o);

mux8 mx0({i0[0],i1[0],i2[0],i3[0],i4[0],i5[0],i6[0],i7[0]},s2,s1,s0,o[0]);

mux8 mx1({i0[1],i1[1],i2[1],i3[1],i4[1],i5[1],i6[1],i7[1]},s2,s1,s0,o[1]);

mux8 mx2({i0[2],i1[2],i2[2],i3[2],i4[2],i5[2],i6[2],i7[2]},s2,s1,s0,o[2]);

mux8 mx3({i0[3],i1[3],i2[3],i3[3],i4[3],i5[3],i6[3],i7[3]},s2,s1,s0,o[3]);

mux8 mx4({i0[4],i1[4],i2[4],i3[4],i4[4],i5[4],i6[4],i7[4]},s2,s1,s0,o[4]);

mux8 mx5({i0[5],i1[5],i2[5],i3[5],i4[5],i5[5],i6[5],i7[5]},s2,s1,s0,o[5]);

mux8 mx6({i0[6],i1[6],i2[6],i3[6],i4[6],i5[6],i6[6],i7[6]},s2,s1,s0,o[6]);

mux8 mx7({i0[7],i1[7],i2[7],i3[7],i4[7],i5[7],i6[7],i7[7]},s2,s1,s0,o[7]);

mux8 mx8({i0[8],i1[8],i2[8],i3[8],i4[8],i5[8],i6[8],i7[8]},s2,s1,s0,o[8]);

mux8 mx9({i0[9],i1[9],i2[9],i3[9],i4[9],i5[9],i6[9],i7[9]},s2,s1,s0,o[9]);

mux8 mx10({i0[10],i1[10],i2[10],i3[10],i4[10],i5[10],i6[10],i7[10]},s2,s1,s0,o[10]);

mux8 mx11({i0[11],i1[11],i2[11],i3[11],i4[11],i5[11],i6[11],i7[11]},s2,s1,s0,o[11]);

mux8 mx12({i0[12],i1[12],i2[12],i3[12],i4[12],i5[12],i6[12],i7[12]},s2,s1,s0,o[12]);

mux8 mx13({i0[13],i1[13],i2[13],i3[13],i4[13],i5[13],i6[13],i7[13]},s2,s1,s0,o[13]);

mux8 mx14({i0[14],i1[14],i2[14],i3[14],i4[14],i5[14],i6[14],i7[14]},s2,s1,s0,o[14]);

mux8 mx15({i0[15],i1[15],i2[15],i3[15],i4[15],i5[15],i6[15],i7[15]},s2,s1,s0,o[15]);

endmodule

module reg\_file (input wire clk, reset, wr, input wire [2:0] rd\_addr\_a, rd\_addr\_b, wr\_addr, input wire [15:0] d\_in, output wire [15:0] d\_out\_a, d\_out\_b);

// Declare wires here

wire [0:7]load;

wire [0:15]r0,r1,r2,r3,r4,r5,r6,r7;

// Instantiate modules here

demux8 d0(wr,wr\_addr[2],wr\_addr[1],wr\_addr[0],load);

dfrl16 reg0(clk,reset,load[0],d\_in,r0);

dfrl16 reg1(clk,reset,load[1],d\_in,r1);

dfrl16 reg2(clk,reset,load[2],d\_in,r2);

dfrl16 reg3(clk,reset,load[3],d\_in,r3);

dfrl16 reg4(clk,reset,load[4],d\_in,r4);

dfrl16 reg5(clk,reset,load[5],d\_in,r5);

dfrl16 reg6(clk,reset,load[6],d\_in,r6);

dfrl16 reg7(clk,reset,load[7],d\_in,r7);

mux128\_16 m0(r0,r1,r2,r3,r4,r5,r6,r7,rd\_addr\_a[0],rd\_addr\_a[1],rd\_addr\_a[2],d\_out\_a);

mux128\_16 m1(r0,r1,r2,r3,r4,r5,r6,r7,rd\_addr\_b[0],rd\_addr\_b[1],rd\_addr\_b[2],d\_out\_b);

endmodule

module mux2\_16(input wire [15:0] din\_regular, alu\_out, input wire i, output wire [15:0]din\_final);

mux2 m0(din\_regular[0], alu\_out[0], i, din\_final[0]);

mux2 m1(din\_regular[1], alu\_out[1], i, din\_final[1]);

mux2 m2(din\_regular[2], alu\_out[2], i, din\_final[2]);

mux2 m3(din\_regular[3], alu\_out[3], i, din\_final[3]);

mux2 m4(din\_regular[4], alu\_out[4], i, din\_final[4]);

mux2 m5(din\_regular[5], alu\_out[5], i, din\_final[5]);

mux2 m6(din\_regular[6], alu\_out[6], i, din\_final[6]);

mux2 m7(din\_regular[7], alu\_out[7], i, din\_final[7]);

mux2 m8(din\_regular[8], alu\_out[8], i, din\_final[8]);

mux2 m9(din\_regular[9], alu\_out[9], i, din\_final[9]);

mux2 m10(din\_regular[10], alu\_out[10], i, din\_final[10]);

mux2 m11(din\_regular[11], alu\_out[11], i, din\_final[11]);

mux2 m12(din\_regular[12], alu\_out[12], i, din\_final[12]);

mux2 m13(din\_regular[13], alu\_out[13], i, din\_final[13]);

mux2 m14(din\_regular[14], alu\_out[14], i, din\_final[14]);

mux2 m15(din\_regular[15], alu\_out[15], i, din\_final[15]);

endmodule

module reg\_alu (input wire clk, reset, sel, wr, input wire [1:0] op, input wire [2:0] rd\_addr\_a, rd\_addr\_b, wr\_addr, input wire [15:0] d\_in, output wire [15:0] d\_out\_a, d\_out\_b, output wire cout);

// Declare wires here

wire [15:0] alu\_out;

wire [15:0] din1;

// Instantiate modules here

mux2\_16 m0(d\_in, alu\_out, sel, din1);

reg\_file reg0(clk, reset, wr, rd\_addr\_a, rd\_addr\_b, wr\_addr, din1, d\_out\_a, d\_out\_b);

alu alu0(op, d\_out\_a, d\_out\_b, alu\_out, cout);

endmodule

**tb\_reg\_alu.v :**

`timescale 1 ns / 100 ps

`define TESTVECS 8

module tb;

reg clk, reset, wr, sel;

reg [1:0] op;

reg [2:0] rd\_addr\_a, rd\_addr\_b, wr\_addr; reg [15:0] d\_in;

wire [15:0] d\_out\_a, d\_out\_b;

reg [28:0] test\_vecs [0:(`TESTVECS-1)];

integer i;

initial begin $dumpfile("tb\_reg\_alu.vcd"); $dumpvars(0,tb); end

initial begin reset = 1'b1; #12.5 reset = 1'b0; end

initial clk = 1'b0; always #5 clk =~ clk;

initial begin

test\_vecs[0][28] = 1'b0; test\_vecs[0][27] = 1'b1; test\_vecs[0][26:25] = 2'b00;

test\_vecs[0][24:22] = 3'o4; test\_vecs[0][21:19] = 3'o3;

test\_vecs[0][18:16] = 3'h3; test\_vecs[0][15:0] = 16'hcdef;

test\_vecs[1][28] = 1'b0; test\_vecs[1][27] = 1'b1; test\_vecs[1][26:25] = 2'b11;

test\_vecs[1][24:22] = 3'o1; test\_vecs[1][21:19] = 3'o5;

test\_vecs[1][18:16] = 3'o7; test\_vecs[1][15:0] = 16'h3210;

test\_vecs[2][28] = 1'b0; test\_vecs[2][27] = 1'b1; test\_vecs[2][26:25] = 2'b10;

test\_vecs[2][24:22] = 3'o3; test\_vecs[2][21:19] = 3'o7;

test\_vecs[2][18:16] = 3'o5; test\_vecs[2][15:0] = 16'h4567;

test\_vecs[3][28] = 1'b0; test\_vecs[3][27] = 1'b1; test\_vecs[3][26:25] = 2'b01;

test\_vecs[3][24:22] = 3'o1; test\_vecs[3][21:19] = 3'o5;

test\_vecs[3][18:16] = 3'o1; test\_vecs[3][15:0] = 16'hba98;

test\_vecs[4][28] = 1'b0; test\_vecs[4][27] = 1'b0; test\_vecs[4][26:25] = 2'b11;

test\_vecs[4][24:22] = 3'o1; test\_vecs[4][21:19] = 3'o5;

test\_vecs[4][18:16] = 3'o1; test\_vecs[4][15:0] = 16'h2345;

test\_vecs[5][28] = 1'b1; test\_vecs[5][27] = 1'b1; test\_vecs[5][26:25] = 2'b00;

test\_vecs[5][24:22] = 3'o1; test\_vecs[5][21:19] = 3'o5;

test\_vecs[5][18:16] = 3'o2; test\_vecs[5][15:0] = 16'hde33;

test\_vecs[6][28] = 1'b1; test\_vecs[6][27] = 1'b1; test\_vecs[6][26:25] = 2'b01;

test\_vecs[6][24:22] = 3'o3; test\_vecs[6][21:19] = 3'o7;

test\_vecs[6][18:16] = 3'o4; test\_vecs[6][15:0] = 16'haa21;

test\_vecs[7][28] = 1'b1; test\_vecs[7][27] = 1'b0; test\_vecs[7][26:25] = 2'b01;

test\_vecs[7][24:22] = 3'o0; test\_vecs[7][21:19] = 3'o0;

test\_vecs[7][18:16] = 3'o2; test\_vecs[7][15:0] = 16'ha341;

end

initial {sel, wr, op, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in} = 0;

reg\_alu reg\_alu\_0 (clk, reset, sel, wr, op, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in,

d\_out\_a, d\_out\_b, cout);

initial begin

#6 for(i=0;i<`TESTVECS;i=i+1)

begin #10 {sel, wr, op, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in}=test\_vecs[i]; end

#100 $finish;

end

endmodule

**Output waveform**



