DDCO WEEK - 6

NANDAN N

PES1UG21CS361

REGISTER-ALU

SOURCE CODE

```
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 mux16 (input wire[15:0] i0,i1,input wire j,output
wire[15:0] o);
  wire [15:0] t;
  mux2 mo_1(i0[0],i1[0],j,o[0]);
  mux2 mo_2(i0[1],i1[1],j,o[1]);
  mux2 mo_3(i0[2],i1[2],j,o[2]);
```

```
  mux2 mo_4(i0[3],i1[3],j,o[3]);
  mux2 mo_5(i0[4],i1[4],j,o[4]);
  mux2 mo_6(i0[5],i1[5],j,o[5]);
  mux2 mo_7(i0[6],i1[6],j,o[6]);
  mux2 mo_8(i0[7],i1[7],j,o[7]);
  mux2 mo_9(i0[8],i1[8],j,o[8]);
  mux2 mo_10(i0[9],i1[9],j,o[9]);
  mux2 mo_11(i0[10],i1[10],j,o[10]);
  mux2 mo_12(i0[11],i1[11],j,o[11]);
  mux2 mo_13(i0[12],i1[12],j,o[12]);
  mux2 mo_14(i0[13],i1[13],j,o[13]);
  mux2 mo_15(i0[14],i1[14],j,o[14]);
  mux2 mo_16(i0[15],i1[15],j,o[15]);
endmodule

module fa (input wire i0, i1, cin, output wire sum, cout);
wire t0, t1, t2;
   xor3 _i0 (i0, i1, cin,sum);
   and2 _i1 (i0, i1, t0);
   and2 _i2 (i1, cin, t1);
   and2 _i3 (cin, i0, t2);
   or3 _i4 (t0, t1,t2, cout);
endmodule

module addsub (input wire addsub, i0, i1, cin, output wire
sumdiff, cout);
wire t;
fa _i0 (i0, t, cin, sumdiff, cout);
  xor2 _i1 (i1, addsub, t);
endmodule

module alu_slice(input wire [1:0] op, input wire i0, i1,
cin, output wire o, cout);
wire t_sumdiff, t_and, t_or, t_andor;
```

```
addsub _i0 (op[0], i0, i1, cin, t_sumdiff, cout);
   and2 _i1 (i0, i1, t_and);
   or2 _i2 (i0, i1, t_or);
   mux2 _i3 (t_and, t_or, op[0], t_andor);
   mux2 _i4 (t_sumdiff, t_andor, 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);
wire[14:0] c;
alu_slice _i0 (op, i0[0], i1[0], op[0] , o[0], c[0]);
alu_slice _i1 (op, i0[1], i1[1], c[0], o[1], c[1]);
alu_slice _i2 (op, i0[2], i1[2], c[1], o[2], c[2]);
alu_slice _i3 (op, i0[3], i1[3], c[2], o[3], c[3]);
alu_slice _i4 (op, i0[4], i1[4], c[3], o[4], c[4]);
alu_slice _i5 (op, i0[5], i1[5], c[4], o[5], c[5]);
alu_slice _i6 (op, i0[6], i1[6], c[5], o[6], c[6]);
alu_slice _i7 (op, i0[7], i1[7], c[6], o[7], c[7]);
alu_slice _i8 (op, i0[8], i1[8], c[7], o[8], c[8]);
alu_slice _i9 (op, i0[9], i1[9], c[8], o[9], c[9]);
alu_slice _i10 (op, i0[10], i1[10], c[9] , o[10], c[10]);
alu_slice _i11 (op, i0[11], i1[11], c[10], o[11], c[11]);
alu_slice _i12 (op, i0[12], i1[12], c[11], o[12], c[12]);
alu_slice _i13 (op, i0[13], i1[13], c[12], o[13], c[13]);
alu_slice _i14 (op, i0[14], i1[14], c[13], o[14], c[14]);
alu_slice _i15 (op, i0[15], i1[15], c[14], o[15], cout);
endmodule

module reg_alu(input wire clk, reset,sel,wr,input
wire[1:0] op,input wire[2:0] rdadder_a,rdadder_b, wr_addr,
input wire[15:0]din, output wire[15:0]dout_a,
dout_b,output cout);
  input wire [15:0] alu_out;
  input wire cout_o;
```

```
  //reg
r1(clk,reset,wr,rdadder_a,rdadder_b,wr_addr,din,dout_a,dou
t_b);
  alu a1(op,dout_a,dout_b,alu_out,cout_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 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
```

TEST BENCH CODE

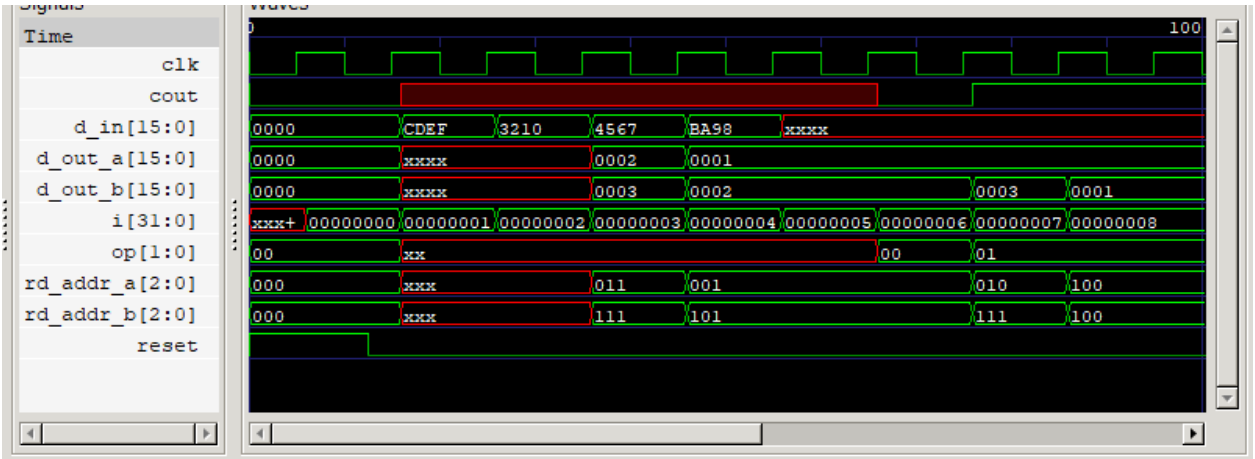
```
`timescale 1 ns / 100 ps
`define TESTVECS 8

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 cout;
  assign cout={1'b0+op[0]}+{1'b0+op[1]};
  assign
d_out_a={1'b0+rd_addr_a[0]}+{1'b0+rd_addr_a[1]}+{1'b0+
rd_addr_a[2]};
  assign
d_out_b={1'b0+rd_addr_b[0]}+{1'b0+rd_addr_b[1]}+{1'b0+
rd_addr_b[2]};

  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'bxx;
    test_vecs[0][24:22] = 3'ox; test_vecs[0][21:19] =
3'ox;
    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'bxx;
    test_vecs[1][24:22] = 3'ox; test_vecs[1][21:19] =
3'ox;
    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'bxx;
    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'bxx;
    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'bxx;
    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'hxxxx;

    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'hxxxx;

    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'o2; test_vecs[6][21:19] =
3'o7;
    test_vecs[6][18:16] = 3'o4; test_vecs[6][15:0] =
16'hxxxx;

    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'o4; test_vecs[7][21:19] =
3'o4;
    test_vecs[7][18:16] = 3'ox; test_vecs[7][15:0] =
16'hxxxx;
  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




```
C:\iverilog\bin>iverilog -o sample lib_reg.v tb_reg_alu.v

C:\iverilog\bin>vvp sample
VCD info: dumpfile tb_reg_alu.vcd opened for output.

C:\iverilog\bin>gtkwave tb_reg_alu.vcd

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

[0] start time.
[186000] end time.
```