Verilog Files and Testbenches

W1

File: ./w1/helloworld.v

```
module main;
  initial
    begin
      $display("Hello World");
      $finish;
    end
endmodule
```

File: ./w1/NOT/NOT.v

~~~
module NOT(input a, output y);
    assign y = !a;
endmodule```

## File: ./w1/NOT/NOT_tb.v

```verilog
module NOT_TEST;
    reg a;
    wire y;
    not NOT_TEST(y, a);
    initial begin
        #0 a = 0;
        #5 a = 1;
    end
    initial begin
        $monitor($time, "a = %b, y = %b", a, y);
    end
    initial begin
        $dumpfile("NOT.vcd");
        $dumpvars(0, NOT_TEST);
    end
endmodule```

## File: ./w1/NAND/NAND_tb.v
~~~

~~~~
```verilog
module NAND_TEST;
    reg a, b;
    wire y;
    nand NAND_TEST(y, a, b);
    initial begin
        #0 a = 0; b = 0;
        #5 a = 0; b = 1;
        #10 a = 1; b = 0;
        #15 a = 1; b = 1;
    end
    initial begin
        $monitor($time, "a = %b, b = %b, y = %b", a, b, y);
    end
    initial begin
        $dumpfile("NAND.vcd");
        $dumpvars(0, NAND_TEST);
    end
endmodule```

## File: ./w1/NAND/NAND.v
```verilog
module NAND(input a, input b, output y);
    assign y = !(a & b);
endmodule```

## File: ./w1/XOR/XOR_tb.v

```verilog
module XOR_TEST;
    reg a, b;
    wire y;
    xor XOR_TEST(y, a, b);
    initial begin
        #0 a = 0; b = 0;
        #5 a = 0; b = 1;
        #10 a = 1; b = 0;
        #15 a = 1; b = 1;
    end
    initial begin
        $monitor($time, "a = %b, b = %b, y = %b", a, b, y);
    end
    initial begin
        $dumpfile("XOR.vcd");
        $dumpvars(0, XOR_TEST);
~~~~

~~~
    end
endmodule```

## File: ./w1/XOR/XOR.v

```verilog
module XOR(input a, input b, output y);
    assign y = a ^ b;
endmodule
~~~

File: ./w1/AND/AND_tb.v

```
module AND_TEST;
    reg a, b;
    wire y;
    and AND_TEST (y, a, b);
    initial begin
        #0 a = 0; b = 0;
        #5 a = 0; b = 1;
        #10 a = 1; b = 0;
        #15 a = 1; b = 1;
    end
    initial begin
        $monitor($time, "a = %b, b = %b, y = %b", a, b, y);
    end
    initial begin
        $dumpfile("AND.vcd");
        $dumpvars(0, AND_TEST);
    end
endmodule
```

File: ./w1/AND/AND.v

```
module AND(input a, input b, output y);
    assign y = a & b;
endmodule
```

File: ./w1/OR/OR.v

```
module OR(input a, input b, output y);
    assign y = a | b;
endmodule
```

File: ./w1/OR/OR_tb.v

```
module OR_TEST;
    reg a, b;
    wire y;
    or OR_TEST (y, a, b);
    initial begin
        #0 a = 0; b = 0;
        #5 a = 0; b = 1;
        #10 a = 1; b = 0;
        #15 a = 1; b = 1;
    end
    initial begin
        $monitor($time, "a = %b, b = %b, y = %b", a, b, y);
    end
    initial begin
        $dumpfile("OR.vcd");
        $dumpvars(0, OR_TEST);
    end
endmodule
```

File: ./w1/NOR/NOR.v

```
module NOR(input a, input b,output y);
    assign y = !(a | b);
endmodule
```

File: ./w1/NOR/NOR_tb.v

```
module NOR_TEST;
    reg a, b;
    wire y;
    nor NOR_TEST(y, a, b);
    initial begin
        #0 a = 0; b = 0;
        #5 a = 0; b = 1;
        #10 a = 1; b = 0;
        #15 a = 1; b = 1;
    end
    initial begin
        $monitor($time, "a = %b, b = %b, y = %b", a, b, y);
    end
    initial begin
```

~~~
        $dumpfile("NOR.vcd");
        $dumpvars(0, NOR_TEST);
    end
endmodule```

## File: ./w1/XNOR/XNOR_tb.v

```verilog
module XNOR_TEST;
    reg a, b;
    wire y;
    xnor XNOR_TEST(y, a, b);
    initial begin
        #0 a = 0; b = 0;
        #5 a = 0; b = 1;
        #10 a = 1; b = 0;
        #15 a = 1; b = 1;
    end
    initial begin
        $monitor($time, "a = %b, b = %b, y = %b", a, b, y);
    end
    initial begin
        $dumpfile("XNOR.vcd");
        $dumpvars(0, XNOR_TEST);
    end
endmodule```

## File: ./w1/XNOR/XNOR.v

```verilog
module XNOR(input a, input b, output y);
    assign y = !(a ^ b);
endmodule```

---
# W2

## File: ./w2/circuit2_tb.v

```verilog
module tb_simple_circuit;

reg A,B,C;

simple_circuit M1(A,B,C,Y);
~~~

```
initial

begin

A=1'b0 ; B=1'b0 ; C=1'b0;
#20
A=1'b0 ; B=1'b0 ; C=1'b1;
#20
A=1'b0 ; B=1'b1 ; C=1'b0;
#20
A=1'b0 ; B=1'b1 ; C=1'b1;
#20
A=1'b1 ; B=1'b0 ; C=1'b0;
#20
A=1'b1 ; B=1'b0 ; C=1'b1;
#20
A=1'b1 ; B=1'b1 ; C=1'b0;
#20
A=1'b1 ; B=1'b1 ; C=1'b1;

end



initial
begin
$monitor($time, "A2=%b, B2=%b, C2=%b,Y=%b", A,B,C,Y);
end
initial
begin
$dumpfile ("circuit2.vcd");
 $dumpvars (0, tb_simple_circuit);
end
endmodule
```

W2

File: ./w2/simple_circuit.v

```
module simple_circuit (A, B, C, D, E);
    input A, B, C;
    output D, E;
    wire w1;
```

~~~
    and G1 (w1, A, B);
    not G2 (E, C);
    or G3 (D, w1, E);
endmodule```

## File: ./w2/circuit1.v

```verilog
module simple_circuit (A, B, C, Y);
    input A, B, C;
    output Y;
    wire w1;
    and G1 (w1, B, C);
    or G2 (Y, A, w1);
endmodule```

## File: ./w2/simple_circuit_tb.v

```verilog
module tb_simple_circuit;

reg A,B,C;
wire D,E;
simple_circuit M1(A,B,C,D,E);

initial

begin
#20
A=1'b0 ; B=1'b0 ; C=1'b0;
#20
A=1'b0 ; B=1'b0 ; C=1'b1;
#20
A=1'b0 ; B=1'b1 ; C=1'b0;
#20
A=1'b0 ; B=1'b1 ; C=1'b1;
#20
A=1'b1 ; B=1'b0 ; C=1'b0;
#20
A=1'b1 ; B=1'b0 ; C=1'b1;
#20
A=1'b1 ; B=1'b1 ; C=1'b0;
#20
A=1'b1 ; B=1'b1 ; C=1'b1;

end
~~~

```
initial
begin
$monitor($time, "A=%b, B=%b, C=%b,D=%b,E=%b", A,B,C,D,E);
end
initial
begin
$dumpfile ("simple.vcd");
 $dumpvars (0, tb_simple_circuit);
end
endmodule
```

File: ./w2/circuit2.v

```
module simple_circuit(A2,B2,C2,Y);
    input A2,B2,C2;
    output Y;
    wire w1,w2,w3;
    and G1 (w1, C, B);
    or G2 (w2, A, w1);
    and G3 (w3, B, A);
    or G4 (Z, w2, w3);
endmodule
```

File: ./w2/circuit1_tb.v

```
module tb_simple_circuit;
reg A,B,C;
wire Y;
simple_circuit M1(A,B,C,Y);
initial
begin
#20
A=1'b0 ; B=1'b0 ; C=1'b0;
#20
A=1'b0 ; B=1'b0 ; C=1'b1;
#20
A=1'b0 ; B=1'b1 ; C=1'b0;
#20
A=1'b0 ; B=1'b1 ; C=1'b1;
#20
A=1'b1 ; B=1'b0 ; C=1'b0;
#20
```

```
A=1'b1 ; B=1'b0 ; C=1'b1;
#20
A=1'b1 ; B=1'b1 ; C=1'b0;
#20
A=1'b1 ; B=1'b1 ; C=1'b1;
end
initial
begin
$monitor($time, "A=%b, B=%b, C=%b,Y=%b", A,B,C,Y);
end
initial
begin
$dumpfile ("circuit1.vcd");
$dumpvars (0, tb_simple_circuit);
end
endmodule
```

W3

File: ./w3/ha/ha_tb.v

```
module halfadder_tb;
    reg aa, bb;
    wire ss, cy;
    halfadder add1 ( aa, bb, ss, cy );
    initial begin
        aa = 1'b0; bb = 1'b0;
        #5 aa = 1'b0; bb = 1'b1;
        #5 aa = 1'b1; bb = 1'b0;
        #5 aa = 1'b1; bb = 1'b1;
    end
    initial begin
        $monitor($time, "a=%b, b=%b,sum=%b,carry=%b", aa, bb, ss, cy);
    end
    initial begin
        $dumpfile("halfadder.vcd");
        $dumpvars(0, halfadder_tb);
    end
endmodule```

---
# W4
```

~~~
## File: ./w4/2/w4-2.v

```verilog
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 fa (
    input wire i0, i1, cin,
    output wire sum, cout
);
    wire t0, t1, t2;
    xor3 x0 (i0, i1, cin, sum);   // XOR gate for sum
    and2 a0 (i0, i1, t0);         // AND gate for carry-out part 1
    and2 a1 (i1, cin, t1);        // AND gate for carry-out part 2
    and2 a2 (cin, i0, t2);        // AND gate for carry-out part 3
    or3 o0 (t0, t1, t2, cout);    // OR gate to combine carry-out
endmodule

module ha (
    input wire a, b,
    output wire sum, cout
);
    xor2 x0 (a, b, sum);    // XOR gate for sum
    and2 a0 (a, b, cout);   // AND gate for carry-out
endmodule

module circuit3 (
    input wire [2:0] i,
    input wire y,
    output wire S1, Cout
);
    wire S0, C0, y_inv;

    fa fa_1 (i[0], i[1], i[2], S0, C0);
    invert inv_1 (y, y_inv);
    ha ha_1 (S0, y_inv, S1, Cout);
endmodule
```

File: ./w3/rca/fa.v

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

    wire t0, t1, t2;

    xor x0 (t0, a, b);
    xor x1 (sum, t0, cin);

    and a0 (t1, a, b);
    and a1 (t2, a, cin);
    and a2 (t3, b, cin);
    or o0 (cout, t1, t2, t3);

endmodule
```

File: ./w3/fa/fa_tb.v

```
module fulladder_tb;
    reg aa, bb, cc;
    wire ss, cy;
    fulladder add1 ( .a(aa), .b(bb), .cin(cc), .sum(ss), .cout(cy) );
    initial begin
        #0 aa = 1'b0; bb = 1'b0; cc = 1'b0;
        #5 aa = 1'b0; bb = 1'b0; cc = 1'b1;
        #5 aa = 1'b0; bb = 1'b0; cc = 1'b1;
        #5 aa = 1'b0; bb = 1'b1; cc = 1'b0;
        #5 aa = 1'b0; bb = 1'b1; cc = 1'b1;
        #5 aa = 1'b1; bb = 1'b0; cc = 1'b0;
        #5 aa = 1'b1; bb = 1'b0; cc = 1'b1;
        #5 aa = 1'b1; bb = 1'b1; cc = 1'b0;
        #5 aa = 1'b1; bb = 1'b1; cc = 1'b1;
    end
    initial begin
        $monitor($time, "a=%b, b= %b, c= %b,sum= %b,carry= %b", aa, bb,
cc, ss, cy);
    end
    initial begin
        $dumpfile("fulladder.vcd");
        $dumpvars(0, fulladder_tb);
    end
```

~~~
endmodule```

## File: ./w3/fa/fa.v

```verilog
module fulladder (
    input wire a, b, cin,
    output wire sum, cout
);

    wire t0, t1, t2;

    xor x0 (t0, a, b);
    xor x1 (sum, t0, cin);

    and a0 (t1, a, b);
    and a1 (t2, a, cin);
    and a2 (t3, b, cin);
    or o0 (cout, t1, t2, t3);

endmodule
~~~

File: ./w3/rca/rca.v

```
module rca(
    input [3:0]a,b,
    input cin,
    output [3:0]sum,
    output c4);

wire c1,c2,c3;      //Carry out of each full adder

full_adder fa0(a[0],b[0],cin,sum[0],c1);
full_adder fa1(a[1],b[1],c1,sum[1],c2);
full_adder fa2(a[2],b[2],c2,sum[2],c3);
full_adder fa3(a[3],b[3],c3,sum[3],c4);
```

File: ./w3/rca/rca_tb.v

```
module rca_tb;
reg [3:0]a,b;
reg cin;
wire [3:0]sum;
wire c4;
```

```
rca uut(a,b,cin,sum,c4);

initial begin
cin = 0;
a = 4'b0110;
b = 4'b1100;
#10
a = 4'b1110;
b = 4'b1000;
#10
a = 4'b0111;
b = 4'b1110;
#10
a = 4'b0010;
b = 4'b1001;
#10
$finish();
end
endmodule
```

File: ./w3/4-1-mux/mux41_tb.v

```
module TB;
    reg [0:3]ii;
    reg s0;reg s1;
    wire yy;
    mux4 newMUX(.i(ii), .j0(s0),.j1(s1),.o(yy));
    initial begin
        ii = 4'b0000;s0=1'b0;s1=1'b0;
        #5      ii = 4'b1000;s0=1'b0;s1=1'b0;
        #5      ii = 4'b0000;s0=1'b0;s1=1'b1;
        #5      ii = 4'b0100;s0=1'b0;s1=1'b1;
        #5      ii = 4'b0000;s0=1'b1;s1=1'b0;
        #5      ii = 4'b0010;s0=1'b1;s1=1'b0;
        #5      ii = 4'b0000;s0=1'b1;s1=1'b1;
        #5      ii = 4'b0001;s0=1'b1;s1=1'b1;
    end
    initial begin
        $monitor("Time = %0t: ii = %b, s0 = %b, s1 = %b, yy = %b",
$time, ii, s0, s1, yy);
    end
    initial begin
        $dumpfile("MUX4_test.vcd");
        $dumpvars(0, TB);
    end
endmodule
```

File: ./w3/4-1-mux/mux41.v

```
module mux4(input wire [0:3] i, input wire j0, j1, output wire o);
    wire [0:1] s;
    assign s = j1 ? {j0, 1'b1} : {j0, 1'b0};

    assign o = (s == 2'b00) ? i[0] :
              (s == 2'b01) ? i[1] :
              (s == 2'b10) ? i[2] :
                            i[3];
endmodule
```

File: ./w3/2-1-mux/mux21_tb.v

```
module TB;
    reg A, B, S;
    wire X;
    mux2 newMUX(.i0(A), .i1(B), .j(S), .o(X));

    initial begin
        S = 1'b0; A = 1'b0; B = 1'b0;
        #5 S = 1'b0; A = 1'b0; B = 1'b1;
        #5 S = 1'b0; A = 1'b1; B = 1'b0;
        #5 S = 1'b0; A = 1'b1; B = 1'b1;
        #5 S = 1'b1; A = 1'b0; B = 1'b0;
        #5 S = 1'b1; A = 1'b0; B = 1'b1;
        #5 S = 1'b1; A = 1'b1; B = 1'b0;
        #5 S = 1'b1; A = 1'b1; B = 1'b1;
    end
    initial begin
        $monitor("Time = %0t: A = %b, B = %b, S = %b, X = %b", $time, A,
B, S, X);
    end
    initial begin
        $dumpfile("MUX2_test.vcd");
        $dumpvars(0,TB);
    end
endmodule
```

File: ./w3/2-1-mux/mux21.v

~~~
module mux2 (
    input wire i0, i1, j,
    output wire o
);
assign o = (j == 0) ? i0 : i1;

endmodule```

## File: ./w3/ha/ha.v

```verilog
module halfadder (
    input wire a, b,
    output wire sum, cout
);

xor x0 (sum, a, b);
and a0 (cout, a, b);
endmodule
~~~

W4

File: ./w4/1/tb1_circuitvec.v

```
`define TESTVECS 6

module tb;
  reg [2:0] i1;
  reg i2;
  wire sum1,cout1;

  reg [3:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("circuit_3.vcd");
  $dumpvars(0,tb);
  end


  initial begin
    test_vecs[0][3:1] = 3'b000;         test_vecs[0][0:0] = 1'b0;
    test_vecs[1][3:1] = 3'b001; test_vecs[1][0:0] = 1'b1;
    test_vecs[2][3:1] = 3'b010; test_vecs[2][0:0] = 1'b0;
    test_vecs[3][3:1] = 3'b011; test_vecs[3][0:0] = 1'b1;
    test_vecs[4][3:1] = 3'b010; test_vecs[4][0:0] = 1'b0;
```

```
    test_vecs[5][3:1] = 3'b011; test_vecs[5][0:0] = 1'b1;
    end


  initial {i1, i2} = 0;
  circuit3 circuit3_0 (i1,i2,sum1, cout1);
  initial begin
       for(i=0;i<`TESTVECS;i=i+1)
       begin #10 {i1,i2}=test_vecs[i];
       end

  end
endmodule
```

File: ./w4/1/circuitvec.v

```
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 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 circuit3 (input wire [0:2] i1,input wire i2,output wire
sum1,cout1);
```

```
wire x1,x2;
fa fa_1(i1[0],i1[1],i1[2],x1,x2);
fa fa_2(x1,x2,i2,sum1,cout1);
endmodule
```

File: ./w4/2/w4.v

```
module fa (
  input wire i0, i1, cin,
  output wire sum, cout
);

assign sum = i0 ^ i1 ^ cin;
assign cout = (i0&i1 | (i1 & cin) | cin & i0);
endmodule

module inverter (
  input wire a,
  output wire y
);

assign y = ~a;
endmodule;

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

assign sum = a^b;
assign cout = a & b;
endmodule

module circuit3 (

  input wire [2:0] i,
  output wire so, cout
);

wire s0, c0, y;
fa full_adder (
  .i0(i[0]),
  .i1(i[1]),
  .cin(i[2]),
  .sum(s0),
  .cout(c0),
);


inverter inv (
  .a(s0),
  .y(y)
);

ha half_adder (
  .a(y),
  .b(c0),
  .sum(so),
  .cout(cout)
);
endmodule
```

File: ./w4/2/w4_tb.v

```
`define TESTVECS 6

module tb;
  reg [2:0] i1;
  reg i2;
  wire sum1,cout1;

  reg [3:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("circuit_3.vcd");
  $dumpvars(0,tb);
  end


  initial begin
    test_vecs[0][3:1] = 3'b000;         test_vecs[0][0:0] = 1'b0;
    test_vecs[1][3:1] = 3'b001; test_vecs[1][0:0] = 1'b1;
    test_vecs[2][3:1] = 3'b010; test_vecs[2][0:0] = 1'b0;
    test_vecs[3][3:1] = 3'b011; test_vecs[3][0:0] = 1'b1;
    test_vecs[4][3:1] = 3'b010; test_vecs[4][0:0] = 1'b0;
    test_vecs[5][3:1] = 3'b011; test_vecs[5][0:0] = 1'b1;
    end
  initial {i1, i2} = 0;
  circuit3 circuit3_0 (i1,i2,sum1, cout1);
  initial begin
       for(i=0;i<`TESTVECS;i=i+1)
       begin #10 {i1,i2}=test_vecs[i];
       end

  end
endmodule
```

File: ./w4/2/tb1_circuitvec.v

```
`define TESTVECS 6

module tb;
  reg [2:0] i1;
  reg i2;
  wire sum1,cout1;

  reg [3:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("circuit_3.vcd");
  $dumpvars(0,tb);
  end


  initial begin
    test_vecs[0][3:1] = 3'b000;         test_vecs[0][0:0] = 1'b0;
    test_vecs[1][3:1] = 3'b001; test_vecs[1][0:0] = 1'b1;
    test_vecs[2][3:1] = 3'b010; test_vecs[2][0:0] = 1'b0;
    test_vecs[3][3:1] = 3'b011; test_vecs[3][0:0] = 1'b1;
    test_vecs[4][3:1] = 3'b010; test_vecs[4][0:0] = 1'b0;
    test_vecs[5][3:1] = 3'b011; test_vecs[5][0:0] = 1'b1;
    end


  initial {i1, i2} = 0;
  circuit3 circuit3_0 (i1,i2,sum1, cout1);
  initial begin
       for(i=0;i<`TESTVECS;i=i+1)
       begin #10 {i1,i2}=test_vecs[i];
       end
  end
endmodule
```

W5

File: ./w5/alu.v

```
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
```

File: ./w5/lib.v

```
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
```

File: ./w5/alu_tb.v

```
`timescale 1 ns / 100 ps
`define TESTVECS 16

module tb;
    reg clk, reset;
    reg [1:0] op;
    reg [15:0] i0, i1;
    wire [15:0] o;
    wire cout;
    reg [33:0] test_vecs[0:(`TESTVECS-1)];
    integer i;

    initial begin
        $dumpfile("aluf.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][33:32] = 2'b00;
        test_vecs[0][31:16] = 16'h0000;
        test_vecs[0][15:0] = 16'h0000;
        test_vecs[1][33:32] = 2'b00;
        test_vecs[1][31:16] = 16'haa55;
        test_vecs[1][15:0] = 16'h55aa;
        test_vecs[2][33:32] = 2'b00;
        test_vecs[2][31:16] = 16'hffff;
        test_vecs[2][15:0] = 16'h0001;
        test_vecs[3][33:32] = 2'b00;
        test_vecs[3][31:16] = 16'h0001;
        test_vecs[3][15:0] = 16'h7fff;
        test_vecs[4][33:32] = 2'b01;
        test_vecs[4][31:16] = 16'h0000;
        test_vecs[4][15:0] = 16'h0000;
        test_vecs[5][33:32] = 2'b01;
        test_vecs[5][31:16] = 16'haa55;
        test_vecs[5][15:0] = 16'h55aa;
        test_vecs[6][33:32] = 2'b01;
        test_vecs[6][31:16] = 16'hffff;
        test_vecs[6][15:0] = 16'h0001;
        test_vecs[7][33:32] = 2'b01;
        test_vecs[7][31:16] = 16'h0001;
        test_vecs[7][15:0] = 16'h7fff;
        test_vecs[8][33:32] = 2'b10;
        test_vecs[8][31:16] = 16'h0000;
        test_vecs[8][15:0] = 16'h0000;
        test_vecs[9][33:32] = 2'b10;
        test_vecs[9][31:16] = 16'haa55;
        test_vecs[9][15:0] = 16'h55aa;
        test_vecs[10][33:32] = 2'b10;
        test_vecs[10][31:16] = 16'hffff;
        test_vecs[10][15:0] = 16'h0001;
        test_vecs[11][33:32] = 2'b10;
        test_vecs[11][31:16] = 16'h0001;
        test_vecs[11][15:0] = 16'h7fff;
        test_vecs[12][33:32] = 2'b11;
        test_vecs[12][31:16] = 16'h0000;
        test_vecs[12][15:0] = 16'h0000;
        test_vecs[13][33:32] = 2'b11;
        test_vecs[13][31:16] = 16'haa55;
        test_vecs[13][15:0] = 16'h55aa;
        test_vecs[14][33:32] = 2'b11;
        test_vecs[14][31:16] = 16'hffff;
        test_vecs[14][15:0] = 16'h0001;
        test_vecs[15][33:32] = 2'b11;
        test_vecs[15][31:16] = 16'h0001;
        test_vecs[15][15:0] = 16'h7fff;
    end

    initial {op, i0, i1} = 0;
    alu alu_0 (
        op, i0, i1, o, cout
    );
    initial begin
        #6
            for (i = 0; i < `TESTVECS; i = i + 1) begin
                #10{op, i0, i1} = test_vecs[i];
            end
        #100 $finish;
    end

endmodule
```

W6

File: ./w6/alu.v

```
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
```

File: ./w6/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'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
```

File: ./w6/reg_alu.v

```
// 16-bit D Flip-Flop with Reset and Load Enable
module dfrl_16 (input wire clk, reset, load, input wire [0:15] in,
output wire [0:15] out);
  dfrl dfrl_0 (clk, reset, load, in[0], out[0]);
  dfrl dfrl_1 (clk, reset, load, in[1], out[1]);
  dfrl dfrl_2 (clk, reset, load, in[2], out[2]);
  dfrl dfrl_3 (clk, reset, load, in[3], out[3]);
  dfrl dfrl_4 (clk, reset, load, in[4], out[4]);
  dfrl dfrl_5 (clk, reset, load, in[5], out[5]);
  dfrl dfrl_6 (clk, reset, load, in[6], out[6]);
  dfrl dfrl_7 (clk, reset, load, in[7], out[7]);
  dfrl dfrl_8 (clk, reset, load, in[8], out[8]);
  dfrl dfrl_9 (clk, reset, load, in[9], out[9]);
  dfrl dfrl_10 (clk, reset, load, in[10], out[10]);
  dfrl dfrl_11 (clk, reset, load, in[11], out[11]);
  dfrl dfrl_12 (clk, reset, load, in[12], out[12]);
  dfrl dfrl_13 (clk, reset, load, in[13], out[13]);
  dfrl dfrl_14 (clk, reset, load, in[14], out[14]);
  dfrl dfrl_15 (clk, reset, load, in[15], out[15]);
endmodule

// 16-bit 2:1 Multiplexer
module mux2_16 (input wire [15:0] i0, i1, input wire j, output wire
[15:0] o);
  mux2 mux2_0 (i0[0], i1[0], j, o[0]);
  mux2 mux2_1 (i0[1], i1[1], j, o[1]);
  mux2 mux2_2 (i0[2], i1[2], j, o[2]);
  mux2 mux2_3 (i0[3], i1[3], j, o[3]);
  mux2 mux2_4 (i0[4], i1[4], j, o[4]);
  mux2 mux2_5 (i0[5], i1[5], j, o[5]);
  mux2 mux2_6 (i0[6], i1[6], j, o[6]);
  mux2 mux2_7 (i0[7], i1[7], j, o[7]);
  mux2 mux2_8 (i0[8], i1[8], j, o[8]);
  mux2 mux2_9 (i0[9], i1[9], j, o[9]);
  mux2 mux2_10 (i0[10], i1[10], j, o[10]);
  mux2 mux2_11 (i0[11], i1[11], j, o[11]);
  mux2 mux2_12 (i0[12], i1[12], j, o[12]);
  mux2 mux2_13 (i0[13], i1[13], j, o[13]);
  mux2 mux2_14 (i0[14], i1[14], j, o[14]);
  mux2 mux2_15 (i0[15], i1[15], j, o[15]);
endmodule

// 16-bit 8:1 Multiplexer
```

```
module mux8_16 (input wire [0:15] i0, i1, i2, i3, i4, i5, i6, i7, input
wire [0:2] j, output wire [0:15] o);
  mux8 mux8_0 ({i0[0], i1[0], i2[0], i3[0], i4[0], i5[0], i6[0], i7[0]},
j[2], j[1], j[0], o[0]);
  mux8 mux8_1 ({i0[1], i1[1], i2[1], i3[1], i4[1], i5[1], i6[1], i7[1]},
j[2], j[1], j[0], o[1]);
  mux8 mux8_2 ({i0[2], i1[2], i2[2], i3[2], i4[2], i5[2], i6[2], i7[2]},
j[2], j[1], j[0], o[2]);
  mux8 mux8_3 ({i0[3], i1[3], i2[3], i3[3], i4[3], i5[3], i6[3], i7[3]},
j[2], j[1], j[0], o[3]);
  mux8 mux8_4 ({i0[4], i1[4], i2[4], i3[4], i4[4], i5[4], i6[4], i7[4]},
j[2], j[1], j[0], o[4]);
  mux8 mux8_5 ({i0[5], i1[5], i2[5], i3[5], i4[5], i5[5], i6[5], i7[5]},
j[2], j[1], j[0], o[5]);
  mux8 mux8_6 ({i0[6], i1[6], i2[6], i3[6], i4[6], i5[6], i6[6], i7[6]},
j[2], j[1], j[0], o[6]);
  mux8 mux8_7 ({i0[7], i1[7], i2[7], i3[7], i4[7], i5[7], i6[7], i7[7]},
j[2], j[1], j[0], o[7]);
  mux8 mux8_8 ({i0[8], i1[8], i2[8], i3[8], i4[8], i5[8], i6[8], i7[8]},
j[2], j[1], j[0], o[8]);
  mux8 mux8_9 ({i0[9], i1[9], i2[9], i3[9], i4[9], i5[9], i6[9], i7[9]},
j[2], j[1], j[0], o[9]);
  mux8 mux8_10 ({i0[10], i1[10], i2[10], i3[10], i4[10], i5[10], i6[10],
i7[10]}, j[2], j[1], j[0], o[10]);
  mux8 mux8_11 ({i0[11], i1[11], i2[11], i3[11], i4[11], i5[11], i6[11],
i7[11]}, j[2], j[1], j[0], o[11]);
  mux8 mux8_12 ({i0[12], i1[12], i2[12], i3[12], i4[12], i5[12], i6[12],
i7[12]}, j[2], j[1], j[0], o[12]);
  mux8 mux8_13 ({i0[13], i1[13], i2[13], i3[13], i4[13], i5[13], i6[13],
i7[13]}, j[2], j[1], j[0], o[13]);
  mux8 mux8_14 ({i0[14], i1[14], i2[14], i3[14], i4[14], i5[14], i6[14],
i7[14]}, j[2], j[1], j[0], o[14]);
  mux8 mux8_15 ({i0[15], i1[15], i2[15], i3[15], i4[15], i5[15], i6[15],
i7[15]}, j[2], j[1], j[0], o[15]);
endmodule

// Register File Module
module reg_file (input wire clk, reset, wr, input wire [0:2] rd_addr_a,
rd_addr_b, wr_addr, input wire [0:15] d_in, output wire [0:15] d_out_a,
d_out_b);
  wire [0:7] load;
  wire [0:15] dout_0, dout_1, dout_2, dout_3, dout_4, dout_5, dout_6,
dout_7;

  // Write enable logic for register 0
  wire wr_en;
  assign wr_en = (wr_addr != 3'b000) ? wr : 1'b0;

  // Demultiplexer for write enable signals
  demux8 demux8_0 (wr_en, wr_addr[2], wr_addr[1], wr_addr[0], load);

  // Register 0 is hardwired to zero
  assign dout_0 = 16'b0;

  // Instantiate registers 1 to 7
  dfrl_16 dfrl_16_1 (clk, reset, load[1], d_in, dout_1);
  dfrl_16 dfrl_16_2 (clk, reset, load[2], d_in, dout_2);
  dfrl_16 dfrl_16_3 (clk, reset, load[3], d_in, dout_3);
  dfrl_16 dfrl_16_4 (clk, reset, load[4], d_in, dout_4);
  dfrl_16 dfrl_16_5 (clk, reset, load[5], d_in, dout_5);
  dfrl_16 dfrl_16_6 (clk, reset, load[6], d_in, dout_6);
  dfrl_16 dfrl_16_7 (clk, reset, load[7], d_in, dout_7);

  // Multiplexers for read ports
  mux8_16 mux8_16_a (dout_0, dout_1, dout_2, dout_3, dout_4, dout_5,
dout_6, dout_7, rd_addr_a, d_out_a);
  mux8_16 mux8_16_b (dout_0, dout_1, dout_2, dout_3, dout_4, dout_5,
dout_6, dout_7, rd_addr_b, d_out_b);
endmodule

// Register File with ALU (Main Module)
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);

  wire [15:0] d_in_alu, d_in_reg;
  wire cout_0;

  // Instantiate ALU
  alu alu_0 (op, d_out_a, d_out_b, d_in_alu, cout_0);

  // Select between data input and ALU output
  mux2_16 mux2_16_0 (d_in, d_in_alu, sel, d_in_reg);

  // Instantiate Register File
  reg_file reg_file_0 (clk, reset, wr, rd_addr_a, rd_addr_b, wr_addr,
d_in_reg, d_out_a, d_out_b);

  // D Flip-Flop for carry out
  dfr dfr_0 (clk, reset, cout_0, cout);
endmodule
```

File: ./w6/reg_file.v

```
module dfrl_16 (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 mux8_16 (input wire [0:15] i0, i1, i2, i3, i4, i5, i6, i7, input
wire [0:2] j, output wire [0:15] o);
  mux8 mux8_0({i0[0], i1[0], i2[0], i3[0], i4[0], i5[0], i6[0], i7[0]},
j[0], j[1], j[2], o[0]);
  mux8 mux8_1({i0[1], i1[1], i2[1], i3[1], i4[1], i5[1], i6[1], i7[1]},
j[0], j[1], j[2], o[1]);
  mux8 mux8_2({i0[2], i1[2], i2[2], i3[2], i4[2], i5[2], i6[2], i7[2]},
j[0], j[1], j[2], o[2]);
  mux8 mux8_3({i0[3], i1[3], i2[3], i3[3], i4[3], i5[3], i6[3], i7[3]},
j[0], j[1], j[2], o[3]);
  mux8 mux8_4({i0[4], i1[4], i2[4], i3[4], i4[4], i5[4], i6[4], i7[4]},
j[0], j[1], j[2], o[4]);
  mux8 mux8_5({i0[5], i1[5], i2[5], i3[5], i4[5], i5[5], i6[5], i7[5]},
j[0], j[1], j[2], o[5]);
  mux8 mux8_6({i0[6], i1[6], i2[6], i3[6], i4[6], i5[6], i6[6], i7[6]},
j[0], j[1], j[2], o[6]);
  mux8 mux8_7({i0[7], i1[7], i2[7], i3[7], i4[7], i5[7], i6[7], i7[7]},
j[0], j[1], j[2], o[7]);
  mux8 mux8_8({i0[8], i1[8], i2[8], i3[8], i4[8], i5[8], i6[8], i7[8]},
j[0], j[1], j[2], o[8]);
  mux8 mux8_9({i0[9], i1[9], i2[9], i3[9], i4[9], i5[9], i6[9], i7[9]},
j[0], j[1], j[2], o[9]);
  mux8 mux8_10({i0[10], i1[10], i2[10], i3[10], i4[10], i5[10], i6[10],
i7[10]}, j[0], j[1], j[2], o[10]);
```

```
  mux8 mux8_11({i0[11], i1[11], i2[11], i3[11], i4[11], i5[11], i6[11],
i7[11]}, j[0], j[1], j[2], o[11]);
  mux8 mux8_12({i0[12], i1[12], i2[12], i3[12], i4[12], i5[12], i6[12],
i7[12]}, j[0], j[1], j[2], o[12]);
  mux8 mux8_13({i0[13], i1[13], i2[13], i3[13], i4[13], i5[13], i6[13],
i7[13]}, j[0], j[1], j[2], o[13]);
  mux8 mux8_14({i0[14], i1[14], i2[14], i3[14], i4[14], i5[14], i6[14],
i7[14]}, j[0], j[1], j[2], o[14]);
  mux8 mux8_15({i0[15], i1[15], i2[15], i3[15], i4[15], i5[15], i6[15],
i7[15]}, j[0], j[1], j[2], o[15]);
endmodule
module reg_file (input wire  clk, reset, wr, input wire [0:2] rd_addr_a,
rd_addr_b, wr_addr, input wire [0:15] d_in, output wire [0:15] d_out_a,
d_out_b);
  wire [0:7] load;
  wire [0:15] dout_0, dout_1, dout_2, dout_3, dout_4, dout_5, dout_6,
dout_7;
  dfrl_16 dfrl_16_0(clk, reset, load[0], d_in, dout_0);
  dfrl_16 dfrl_16_1(clk, reset, load[1], d_in, dout_1);
  dfrl_16 dfrl_16_2(clk, reset, load[2], d_in, dout_2);
  dfrl_16 dfrl_16_3(clk, reset, load[3], d_in, dout_3);
  dfrl_16 dfrl_16_4(clk, reset, load[4], d_in, dout_4);
  dfrl_16 dfrl_16_5(clk, reset, load[5], d_in, dout_5);
  dfrl_16 dfrl_16_6(clk, reset, load[6], d_in, dout_6);
  dfrl_16 dfrl_16_7(clk, reset, load[7], d_in, dout_7);
  demux8 demux8_0(wr, wr_addr[2], wr_addr[1], wr_addr[0], load);
  mux8_16 mux8_16_9(dout_0, dout_1, dout_2, dout_3, dout_4, dout_5,
dout_6, dout_7, rd_addr_a, d_out_a);
  mux8_16 mux8_16_10(dout_0, dout_1, dout_2, dout_3, dout_4, dout_5,
dout_6, dout_7, rd_addr_b, d_out_b);
endmodule
```

File: ./w6/lib.v

```
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 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
```

W7

File: ./w7/pc.v

```
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 pc_slice (input wire clk, reset, cin, load, inc, sub, offset,
  output wire cout, pc);

 wire in, inc_;

invert invert_0 (inc, inc_);

 and2 and2_0 (offset, inc_, t);

addsub addsub_0 (sub, pc, t, cin, in, cout);

 dfrl dfrl_0 (clk, reset, load, in, pc);

endmodule



module pc_slice0 (input wire clk, reset, cin, load, inc, sub, offset,
output wire cout, pc);

wire in;

 or2 or2_0 (offset, inc, t);

 addsub addsub_0 (sub, pc, t, cin, in, cout);

dfrl dfrl_0 (clk, reset, load, in, pc);

endmodule



module pc (input wire clk, reset, inc, add, sub, input wire [15:0]
offset, output wire [15:0] pc);
```

```
 input wire load;
 input wire [15:0] c;

 or3 or3_0 (inc, add, sub, load);

pc_slice0 pc_slice_0 (clk, reset, sub, load, inc, sub, offset[0], c[0],
pc[0]);

 pc_slice pc_slice_1 (clk, reset, c[0], load, inc, sub, offset[1], c[1],
pc[1]);

pc_slice pc_slice_2 (clk, reset, c[1], load, inc, sub, offset[2], c[2],
pc[2]);

 pc_slice pc_slice_3 (clk, reset, c[2], load, inc, sub, offset[3], c[3],
pc[3]);

  pc_slice pc_slice_4 (clk, reset, c[3], load, inc, sub, offset[4],
c[4], pc[4]);

 pc_slice pc_slice_5 (clk, reset, c[4], load, inc, sub, offset[5], c[5],
pc[5]);

 pc_slice pc_slice_6 (clk, reset, c[5], load, inc, sub, offset[6], c[6],
pc[6]);

 pc_slice pc_slice_7 (clk, reset, c[6], load, inc, sub, offset[7], c[7],
pc[7]);

pc_slice pc_slice_8 (clk, reset, c[7], load, inc, sub, offset[8], c[8],
pc[8]);

  pc_slice pc_slice_9 (clk, reset, c[8], load, inc, sub, offset[9],
c[9], pc[9]);

pc_slice pc_slice_10 (clk, reset, c[9], load, inc, sub, offset[10],
c[10], pc[10]);

pc_slice pc_slice_11 (clk, reset, c[10], load, inc, sub, offset[11],
c[11], pc[11]);

pc_slice pc_slice_12 (clk, reset, c[11], load, inc, sub, offset[12],
c[12], pc[12]);
  pc_slice pc_slice_13 (clk, reset, c[12], load, inc, sub, offset[13],
c[13], pc[13]);
```

```
  pc_slice pc_slice_14 (clk, reset, c[13], load, inc, sub, offset[14],
c[14], pc[14]);
  pc_slice pc_slice_15 (clk, reset, c[14], load, inc, sub, offset[15],
c[15], pc[15]);
endmodule
```

File: ./w7/lib.v

```
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 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
```

File: ./w7/tb_pc.v

```
//   Test bench for PC:

`timescale 1 ns / 100 ps
`define TESTVECS 5

module tb;
  reg clk, reset, inc, add, sub;
  reg [15:0] offset;
  wire [15:0] pc;
  reg [18:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("tb_pc.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][18] = 1'b1; test_vecs[0][17] = 1'b0; test_vecs[0][16] =
1'b0;
    test_vecs[0][15:0] = 15'hxx;
    test_vecs[1][18] = 1'b0; test_vecs[1][17] = 1'b1; test_vecs[1][16] =
1'b0;
    test_vecs[1][15:0] = 15'ha5;
    test_vecs[2][18] = 1'b0; test_vecs[2][17] = 1'b0; test_vecs[2][16] =
1'b0;
    test_vecs[2][15:0] = 15'hxx;
    test_vecs[3][18] = 1'b1; test_vecs[3][17] = 1'b0; test_vecs[3][16] =
1'b0;
    test_vecs[3][15:0] = 15'hxx;
    test_vecs[4][18] = 1'b0; test_vecs[4][17] = 1'b0; test_vecs[4][16] =
1'b1;
    test_vecs[4][15:0] = 15'h14;
  end
  initial {inc, add, sub, offset} = 0;
  pc pc_0 (clk, reset, inc, add, sub, offset, pc);
  initial begin
    #6 for(i=0;i<`TESTVECS;i=i+1)
      begin #10 {inc, add, sub, offset}=test_vecs[i]; end
    #100 $finish;
  end
always@(reset or inc or add or sub )
$monitor("At time = %t, Reset= %b,inc=%b, add=%b,sub = %b,pc =%h ",
$time,reset,inc,add,sub,pc);

endmodule
```