

VERILOG CODE

MODULE 1

alu_1bit module code

```
module alu_1bit(  
    input a1,  
    input b1,  
    input less,  
    input [3:0] op,  
    input carryin,  
    output carryout,  
    output result  
);  
  
not (b1_,b1);  
not (a1_,a1);  
  
mux_2_to_1_I_or_D muxa(a1,a1_,op[3],aout);  
  
mux_2_to_1_I_or_D muxb(b1,b1_,op[2],bout);  
and (andout,bout,aout);  
or (orout,bout,aout);  
  
full_adder fa(a1,b1,carryin,adderout,carryout);  
  
mux_4_to_1 mux4(andout,orout,adderout,less,op[1:0],result);  
  
endmodule
```

alu_1bit test code

```
module alu_1bit1_test;  
  
    // Inputs  
    reg a1;  
    reg b1;  
    reg less;  
    reg [3:0] op;  
    reg carryin;  
  
    // Outputs  
    wire carryout;  
    wire result;  
  
    // Instantiate the Unit Under Test (UUT)  
    alu_1bit uut (  
        .a1(a1),  
        .b1(b1),  
        .less(less),  
        .op(op),  
        .carryin(carryin),  
        .carryout(carryout),  
        .result(result)  
    );  
  
    initial begin
```

```

// Initialize Inputs
a1 = 0;
b1 = 0;
carryin = 0;
op = 0;
less = 0;

// Wait 100 ns for global reset to finish
#100;
a1 = 1;
b1 = 1;
carryin = 1;

#10; op = 0;
#10; op = 1;
#10; op = 2;
#10; op = 3;
#10; op = 7;

#100;
a1 = 1;
b1 = 0;
carryin = 1;

#10; op = 0;
#10; op = 1;
#10; op = 2;
#10; op = 3;
#10; op = 7;

// Add stimulus here

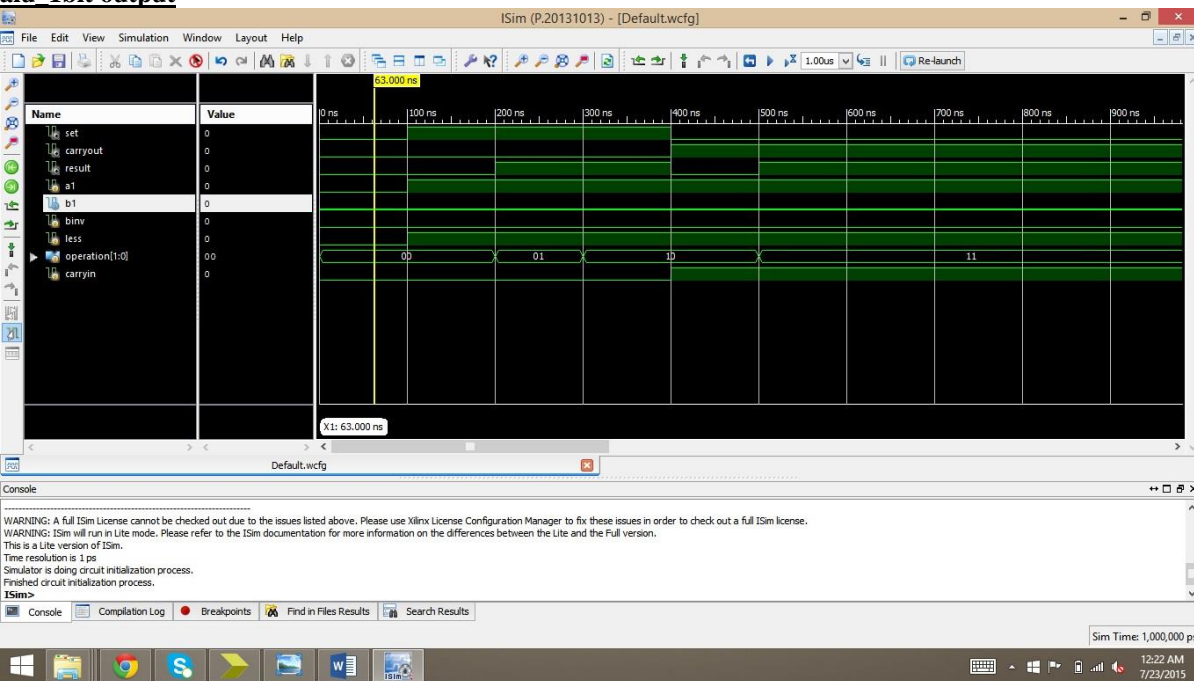
end

```

```

endmodule
alu_1bit output

```



MODULE 2

alu_1bitmsb module code

```

module alu_1bitmsb(

```

```

    input a1,

```

```

input b1,
input less,
input [3:0] op,
input carryin,
    output set,
output carryout,
output result,

    output ov
);

not (b1_,b1);
not (a1_,a1);

mux_2_to_1_I_or_D muxa(a1,a1_,op[3],aout);
mux_2_to_1_I_or_D muxb(b1,b1_,op[2],bout);
and (andout,bout,aout);
or (orout,bout,aout);

full_adder fa(a1,b1,carryin,set,carryout);

mux_4_to_1 mux4(andout,orout,set,less,op[1:0],result);

overflow of(a1,b1,carryin,ov);

endmodule

alu_1bitmsb test code
module alu_1bitmsb1;

    // Inputs
    reg a1;
    reg b1;
    reg less;
    reg [3:0] op;
    reg carryin;

    // Outputs
    wire carryout;
    wire result;
    wire ov;
    wire set;

    // Instantiate the Unit Under Test (UUT)
    alu_1bitmsb uut (
        .a1(a1),
        .b1(b1),
        .less(less),
        .op(op),
        .carryin(carryin),
        .set(set),
        .carryout(carryout),
        .result(result),
        .ov(ov)
    );

    initial begin
        // Initialize Inputs
        a1 = 0;
        b1 = 0;
        less = 0;
        op = 0;
        carryin = 0;

        // Wait 100 ns for global reset to finish

```

```

#100;
a1 = 0;
b1 = 0;
carryin = 1;

#10; op = 0;
#10; op = 1;
#10; op = 2;
#10; op = 3;
#10; op = 7;

#100;
a1 = 1;
b1 = 0;
carryin = 1;

#10; op = 0;
#10; op = 1;
#10; op = 2;
#10; op = 3;
#10; op = 7;

// Add stimulus here

end

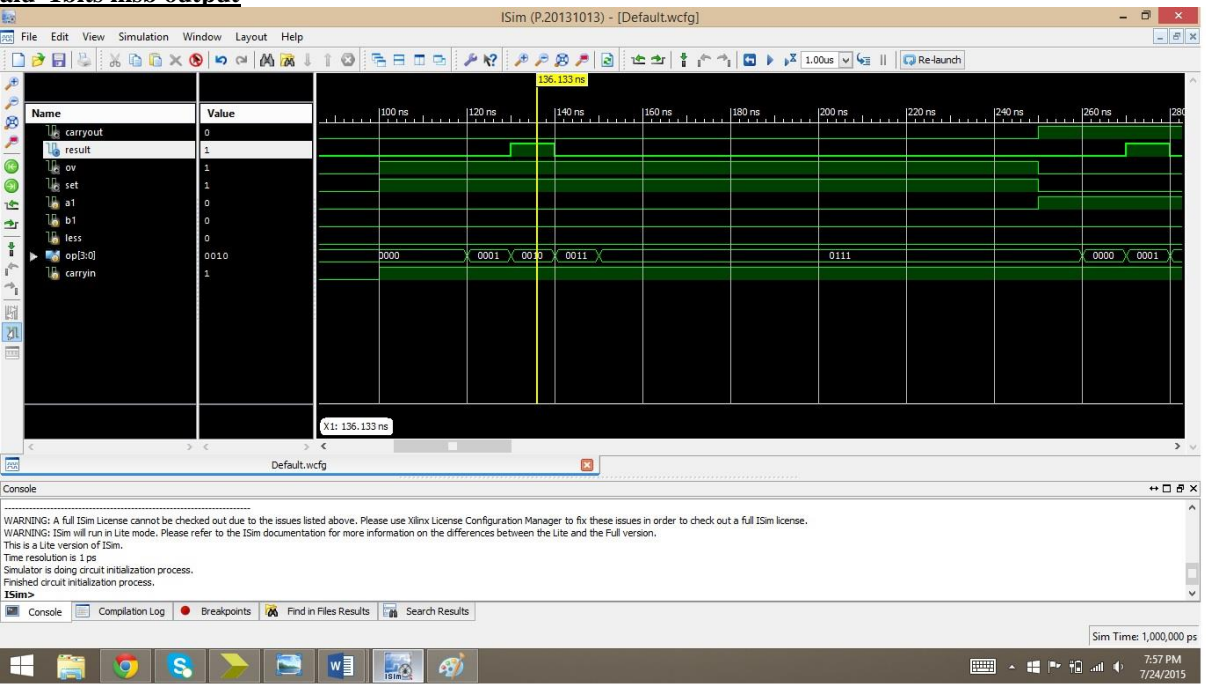
```

```

endmodule

alu_1bits msb output

```



MODULE 3

alu_16bit module code

```

module alu_16bit(
    input [15:0] a1,
    input [15:0] b1,
    input [3:0] operation,
    output [15:0] result,
    output c,
    output overflow,
    output setv,
    output zero
);

```

```

alu_1bit alu0(a1[0],b1[0],setv,operation,1'b0 ,c_0,result[0]),

```

```

        alu1(a1[1],b1[1],1'b0,operation,c_0,c_1,result[1]),
        alu2(a1[2],b1[2],1'b0,operation,c_1,c_2,result[2]),
        alu3(a1[3],b1[3],1'b0,operation,c_2,c_3,result[3]),
        alu4(a1[4],b1[4],1'b0,operation,c_3,c_4,result[4]),
        alu5(a1[5],b1[5],1'b0,operation,c_4,c_5,result[5]),
        alu6(a1[6],b1[6],1'b0,operation,c_5,c_6,result[6]),
        alu7(a1[7],b1[7],1'b0,operation,c_6,c_7,result[7]),
        alu8(a1[8],b1[8],1'b0,operation,c_7,c_8,result[8]),
        alu9(a1[9],b1[9],1'b0,operation,c_8,c_9,result[9]),
    alu10(a1[10],b1[10],1'b0,operation,c_9,c_10,result[10]),
        alu11(a1[11],b1[11],1'b0,operation,c_10,c_11,result[11]),
        alu12(a1[12],b1[12],1'b0,operation,c_11,c_12,result[12]),
    alu13(a1[13],b1[13],1'b0,operation,c_12,c_13,result[13]),
    alu14(a1[14],b1[14],1'b0,operation,c_13,c_14,result[14]);

// calling alu1bit msb
alu_1bitmsb alu15(a1[15],b1[15],1'b0,operation,c_14,setv,c,result[15],overflow);
not (set_,set);
mux_2_to_1_I_or_D setless(set,set_,overflow,setv);
not(zero, zero_);
or ( zero_,result[0], result[1], result[2], result[3],result[4], result[5], result[6], result[7], result[8],
result[9],result[10], result[11], result[12], result[13], result[14], result[15]);

```

endmodule

alu_16bit test code

```

module alu_16bit_test;

    // Inputs
    reg [15:0] a1;
    reg [15:0] b1;
    reg [3:0] operation;

    // Outputs
    wire [15:0] result;
    wire c;
    wire overflow;
    wire setv;
    wire zero;

    // Instantiate the Unit Under Test (UUT)
    alu_16bit uut (
        .a1(a1),
        .b1(b1),
        .operation(operation),
        .result(result),
        .c(c),
        .overflow(overflow),
        .setv(setv),
        .zero(zero)
    );

    initial begin
        // Initialize Inputs
        a1 = 16'h0000;
        b1 = 16'h0001;
        operation = 4'b0000;

        // Wait 100 ns for global reset to finish
        #100;

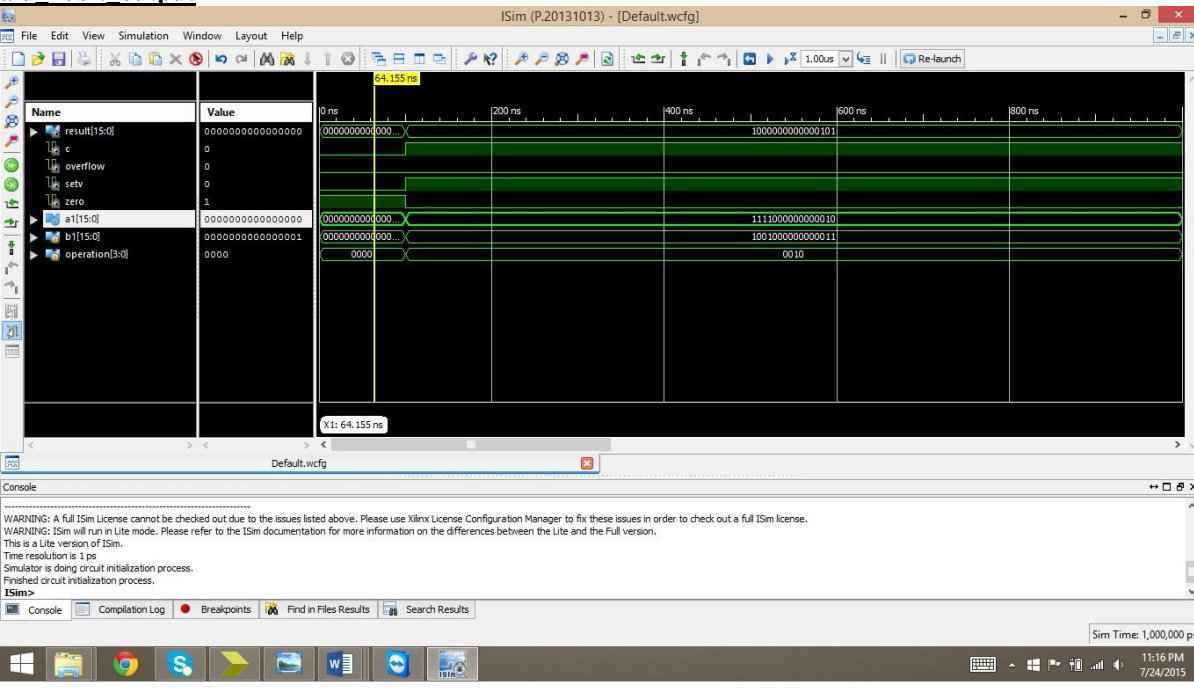
        a1 = 16'hf002;
        b1 = 16'h9003;
        operation = 4'b0010;
        // Add stimulus here
    end

```

end

endmodule

alu 16bit output



MODULE 4

alu_controller module code

```
module alu_controller(  
    input [3:0] aluop,  
    input [1:0] func,  
    output reg [2:0] op  
);  
always@(aluop or func)  
case(aluop)  
4'b0000: //rtype1  
begin  
    if(func==2'b00) op=000;  
    else if(func==2'b01) op=001;  
    else if(func==2'b10) op=010;  
    else if(func==2'b11) op=011;  
end  
4'b0001: //rtype2  
begin  
    if(func==2'b00) op=000;  
    else if(func==2'b01) op=001;  
    else if(func==2'b10) op=010;  
    else if(func==2'b11) op=011;  
end  
4'b0010:  
op=000; //jr  
4'b0100:  
op=001; //lw,sw,il,ibu,sb,addiu,addi,addi_l,addiu_l  
4'b0101:  
op=010; //beqz bne  
4'b0110:  
op=011; //sli,slti,slti_l,sltiu_l
```

```
4'b0111:
op=111;//andi,andi_1
```

```
4'b1001:
op=100;//ori,ori_1
```

```
endcase
endmodule
```

alu_controller test code

```
module alu_controller_test;

    // Inputs
    reg [3:0] aluop;
    reg [1:0] func;

    // Outputs
    wire [2:0] op;

    // Instantiate the Unit Under Test (UUT)
    alu_controller uut (
        .aluop(aluop),
        .func(func),
        .op(op)
    );

    initial begin
        // Initialize Inputs
        aluop = 0;
        func = 0;

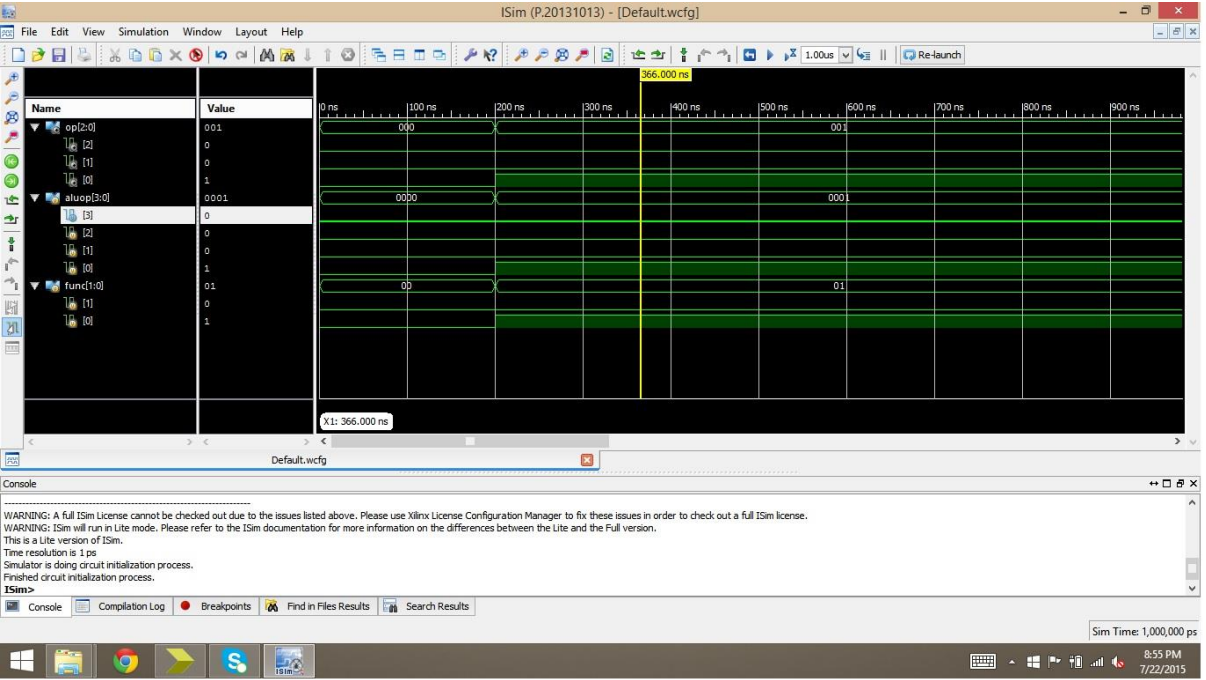
        // Wait 100 ns for global reset to finish
        #100;
        aluop=4'b0000;
        func=2'b00;
        #100;
        aluop=4'b0001;
        func=2'b01;

        // Add stimulus here

    end

endmodule
```

alu_controller output



MODULE 5

Clock module code

```
module clock(
    output reg clkout
);
    initial clkout=0;
    always
    #50
    clkout=~clkout;
```

```
endmodule
```

Clock module test code

```
module clock_test;

    // Outputs
    wire clkout;

    // Instantiate the Unit Under Test (UUT)
    clock uut (
        .clkout(clkout)
    );

    initial begin
        // Initialize Inputs

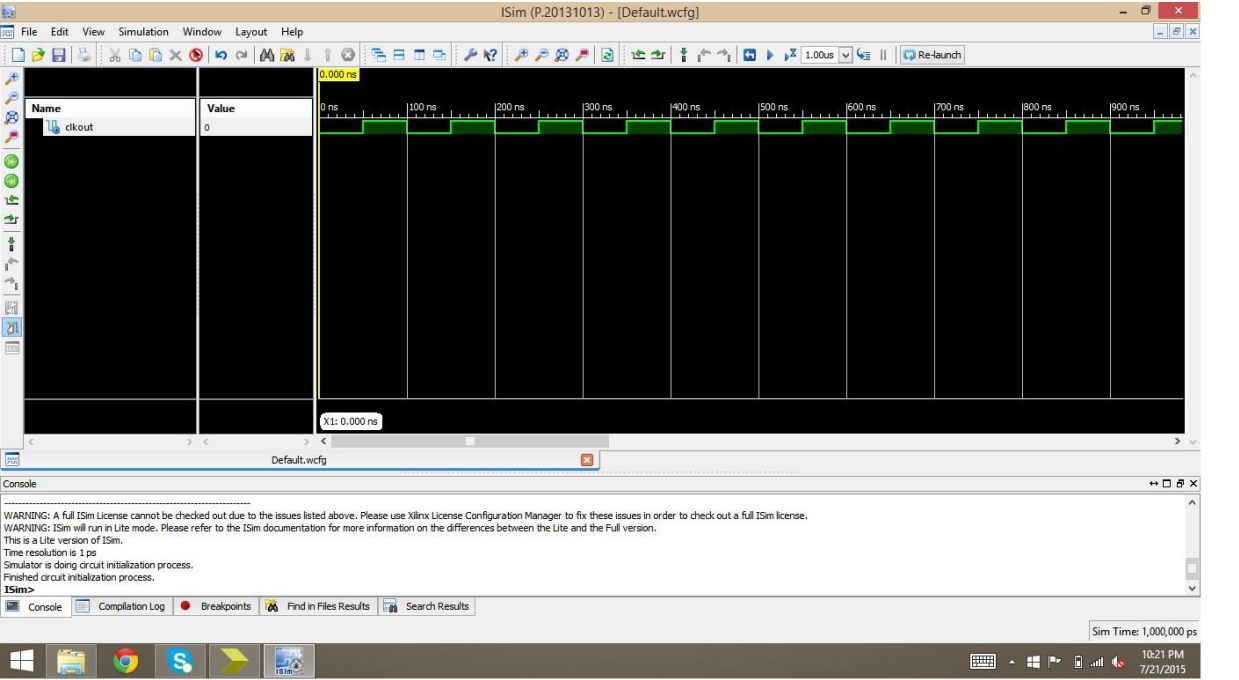
        // Wait 100 ns for global reset to finish
        #100;
        #100;
        #100;
        #100;

        // Add stimulus here

    end

endmodule
```

Clock Output



MODULE 6

Full adder module code

```
module full_adder(
    input faa,
    input fab,
    input cin,
    output r,
    output cout
);
    xor(y0,faa,fab),
                                (r,y0,cin);
    and(y1,y0,cin),
                                (y2,faa,fab);
    or(cout,y1,y2);
```

endmodule

Full adder test code

```
module full_adder_test;
```

```
    // Inputs
```

```
    reg faa;
```

```
    reg fab;
```

```
    reg cin;
```

```
    // Outputs
```

```
    wire r;
```

```
    wire cout;
```

```
    // Instantiate the Unit Under Test (UUT)
```

```
    full_adder uut (
```

```
        .faa(faa),
```

```
        .fab(fab),
```

```
        .cin(cin),
```

```
        .r(r),
```

```
        .cout(cout)
```

```
    );
```

```
    initial begin
```

```
        // Initialize Inputs
```

```
        faa = 0;
```

```

fab = 0;
cin = 0;

// Wait 100 ns for global reset to finish
#100;
cin=1;
#100;
fab=1;cin=0;
#100;
cin=1;
#100;
faa=1;fab=0;cin=0;
#100;
cin=1;
#100;
fab=1;cin=0;
#100;
cin=1;

```

```

// Add stimulus here

```

```

end

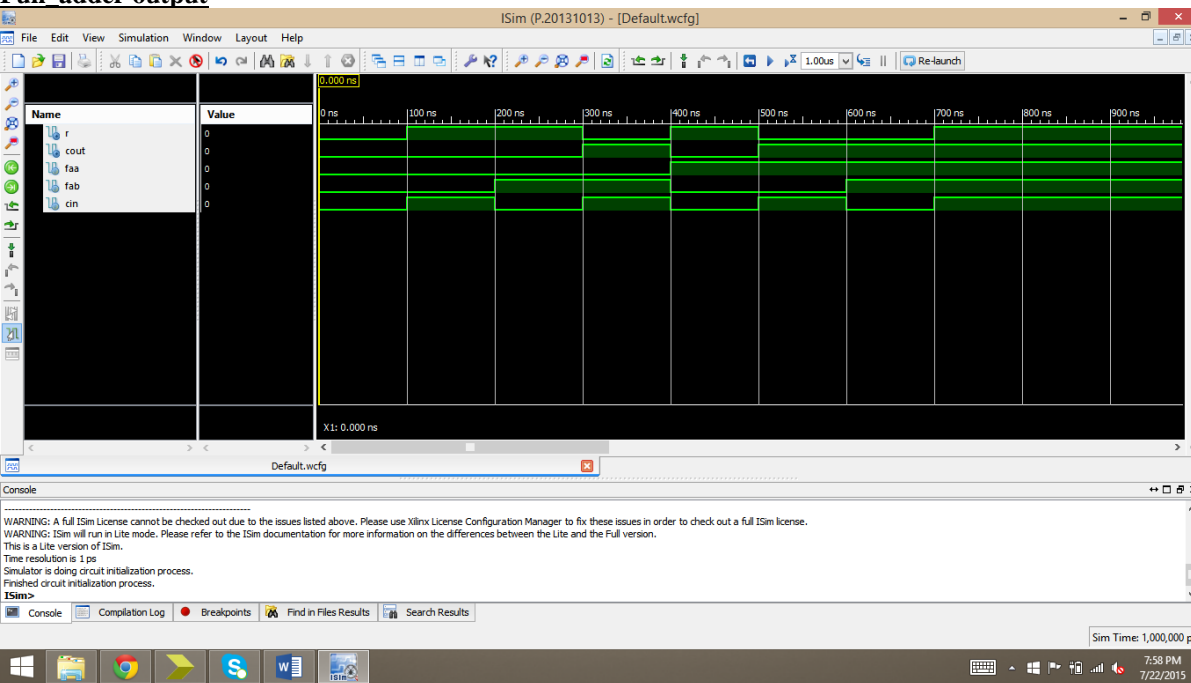
```

```

endmodule

```

Full adder output



MODULE 7

Ir reg module code

```

module ir_reg(
    input [15:0] md,
    output [15:0] inst,
    input irwrite,
    output [10:8] rs,
    output [4:2] rd,
    output [7:5] rt,
    output [10:0] addr,
    output [4:0] imm,
    output [7:0] imm_1,
    output [15:11] opcode,
    output [1:0] func
);
    clock man(clkout);

```

```

reg[15:0] ir_regfile;
reg[15:11] opcode;
reg[10:8] rs;
reg[7:5] rt;
reg[4:2] rd;
reg[1:0] func;
reg[4:0] imm;
reg[7:0] imm_l;
reg[10:0] addr;
reg[15:0] inst;
always@(posedge clkout)

```

```

if(irwrite==1)
begin
ir_regfile=md;
inst=ir_regfile;
opcode=ir_regfile[15:11];
rs=ir_regfile[10:8];
rt=ir_regfile[7:5];
rd=ir_regfile[4:2];
func=ir_regfile[1:0];
imm=ir_regfile[4:0];
imm_l=ir_regfile[7:0];
addr=ir_regfile[10:0];
end
endmodule

```

ir_reg test code

```

module ir_reg_test;

```

```

// Inputs
reg [15:0] md;
reg irwrite;

```

```

// Outputs
wire [15:0] inst;
wire [10:8] rs;
wire [4:2] rd;
wire [7:5] rt;
wire [10:0] addr;
wire [4:0] imm;
wire [7:0] imm_l;
wire [15:11] opcode;
wire [1:0] func;

```

```

// Instantiate the Unit Under Test (UUT)

```

```

ir_reg uut (
    .md(md),
    .inst(inst),
    .irwrite(irwrite),
    .rs(rs),
    .rd(rd),
    .rt(rt),
    .addr(addr),
    .imm(imm),
    .imm_l(imm_l),
    .opcode(opcode),
    .func(func)

```

```

);

```

```

initial begin

```

```

// Initialize Inputs

```

```

md = 16'h0000;

```

```

irwrite = 1;

```

```

// Wait 100 ns for global reset to finish

```

```

#100;

```

```

md=16'hffff;

```

```

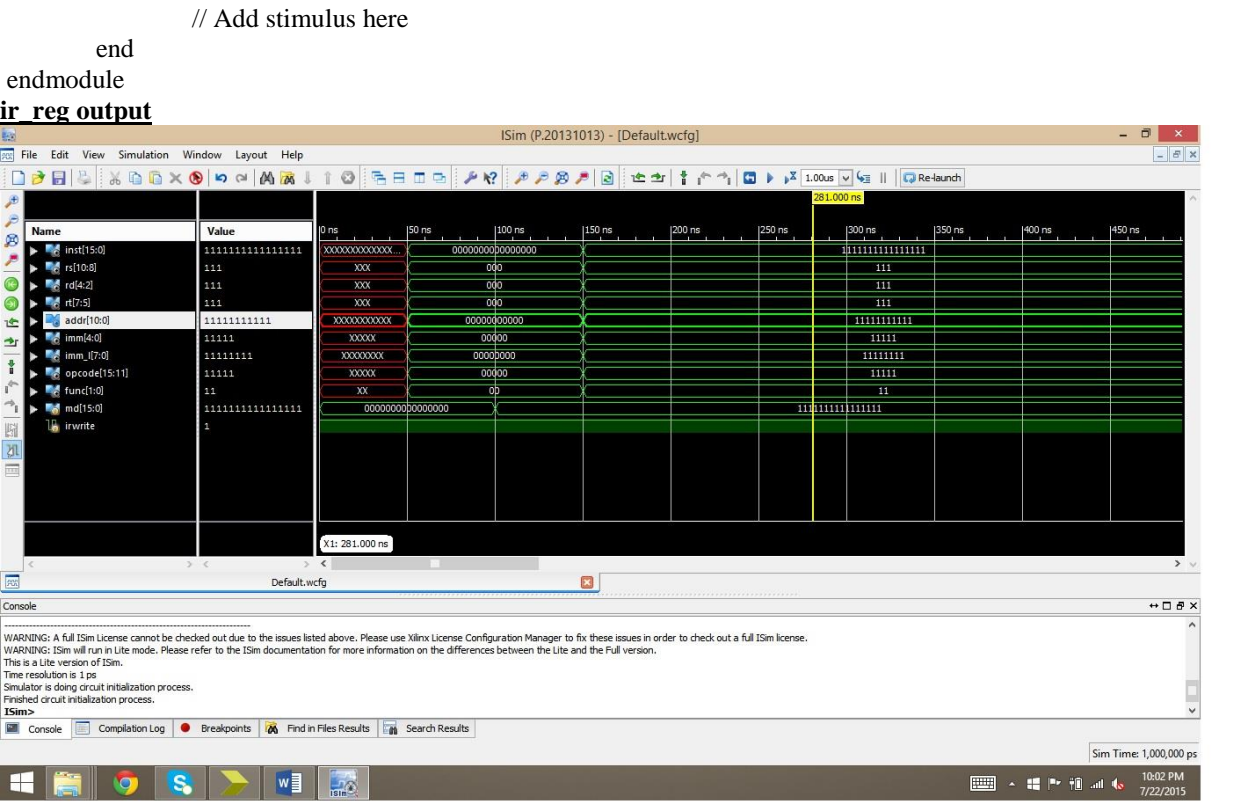
#100;

```

```

irwrite=1;

```



MODULE 8

Jump module code

```
module jump(
    input [15:0] pc,
    input [10:0] address,
    output [15:0] jump_add
);
    assign jump_add[15:0]={pc[3:0],address[10:0],1'b0};
    // Instantiate the Unit Under Test (UUT)
    jump uut (
        .pc(pc),
        .address(address),
        .jump_add(jump_add)
    );

    initial begin
        // Initialize Inputs
        pc = 0;
        address = 0;

        // Wait 100 ns for global reset to finish
        #100;
        pc=15'b00000000000000100;
        address=10'b00000000001;

        #100;

        pc=15'b10000000000000100;
        address=10'b10000000001;
        // Add stimulus here

    end
end
```

endmodule

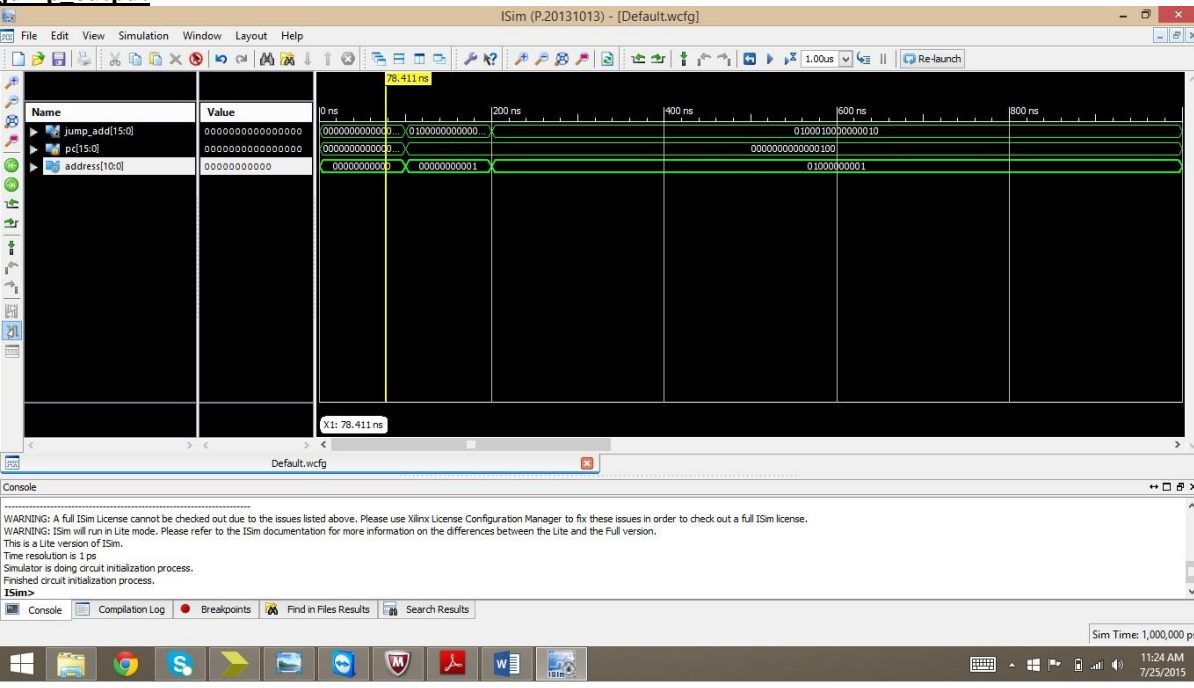
jump test code

```
module jump_test;
```

```
    // Inputs
    reg [15:0] pc;
    reg [10:0] address;
```

```
// Outputs
wire [15:0] jump_add;

jump output
```



```
endmodule
```

MODULE 9

mdr_reg module code

```
module mdr_reg(
    input [15:0] md,
    output [15:0] oumdr
);
    reg[15:0] mdr;
    assign oumdr=md;
    always @(posedge clkout)
        mdr=md;
    clock antman(clkout);
```

```
Endmodule
```

mdr_reg test code

```
module mdr_reg_test;

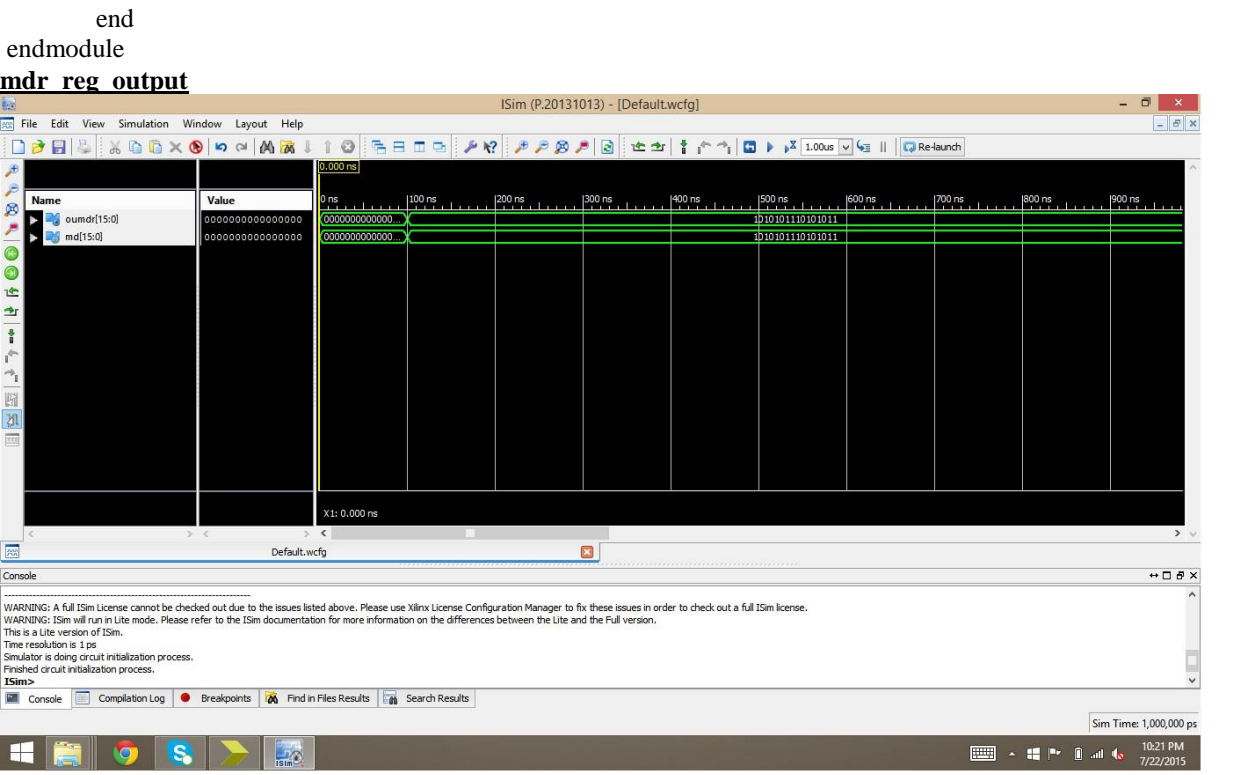
    // Inputs
    reg [15:0] md;

    // Outputs
    wire [15:0] oumdr;

    // Instantiate the Unit Under Test (UUT)
    mdr_reg uut (
        .md(md),
        .oumdr(oumdr)
    );
    initial begin
        // Initialize Inputs
        md = 0;

        // Wait 100 ns for global reset to finish
        #100;
        md=16'hbabab;

        // Add stimulus here
```



MODULE 10

mux 4 to 1 module code

```
module mux_4_to_1(
    input a,
    input b,
    input c,
    input d,
    input [1:0] s,
    output y
);
    assign y= s[1]?(s[0]?d:c):(s[0]?b:a);
```

```
endmodule
```

mux 4 to 1 test code

```
module mux_4_to_1_test;
```

```
    // Inputs
    reg a;
    reg b;
    reg c;
    reg d;
    reg [1:0] s;
```

```
    // Outputs
    wire y;
```

```
    // Instantiate the Unit Under Test (UUT)
```

```
    mux_4_to_1 uut (
        .a(a),
        .b(b),
        .c(c),
        .d(d),
        .s(s),
        .y(y)
    );
```

```
    initial begin
```

```
        // Initialize Inputs
```

```
        a = 0;
```

```

b = 0;
c = 0;
d = 0;
s = 0;

```

```

// Wait 100 ns for global reset to finish

```

```

#100;
a = 1;
#100;
s=2'b01;
#100;
b = 1;
#100;
s=2'b10;
#100;
c = 1;
#100;
s =2'b11;
#100
d=1;

```

```

// Add stimulus here

```

```

end

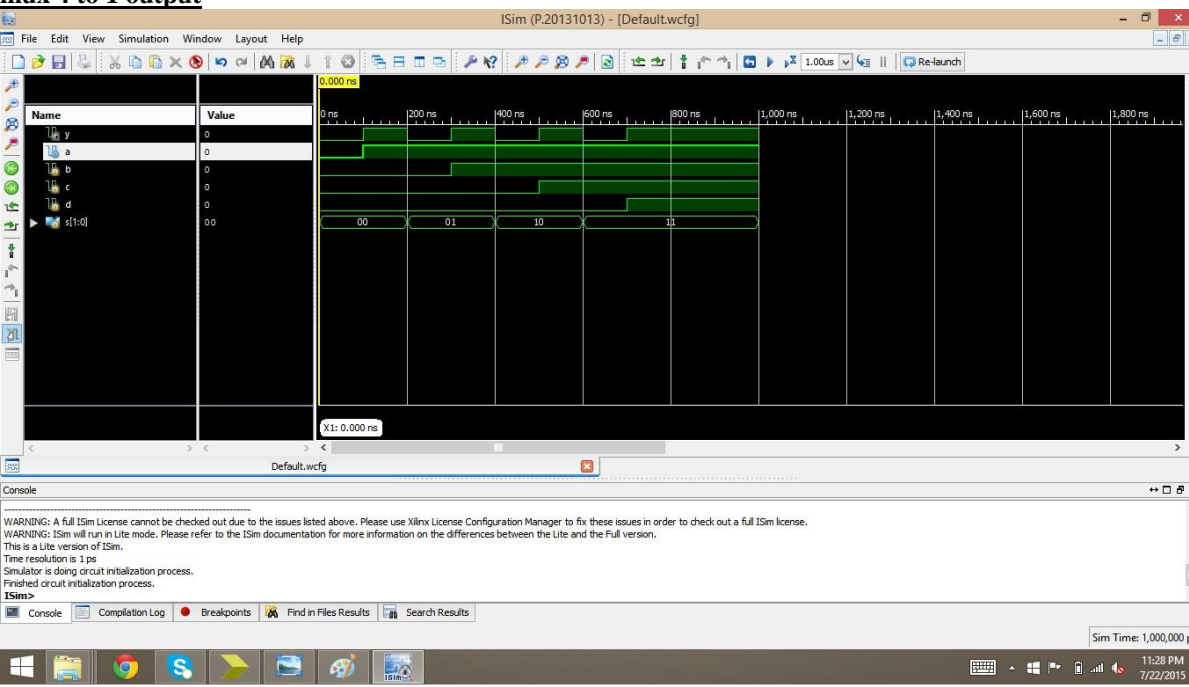
```

```

endmodule

```

mux 4 to 1 output



MODULE 11

Pc module code

```

module PC(
    input clkout,
    input pcwrite,
    input [15:0] inpc,
    output [15:0] oupc
);
    reg [15:0] oupc;
    initial
        oupc=0;
    always@(posedge clkout)
    begin if(pcwrite==1)
        oupc=inpc;
    end

```

endmodule

Pc test code

```
module PC_test;
```

```
    // Inputs
    reg clkout;
    reg pcwrite;
    reg [15:0] inpc;
```

```
    // Outputs
    wire [15:0] oupc;
```

```
    // Instantiate the Unit Under Test (UUT)
```

```
    PC uut (
        .clkout(clkout),
        .pcwrite(pcwrite),
        .inpc(inpc),
        .oupc(oupc)
    );
```

```
    initial begin
```

```
        // Initialize Inputs
        clkout = 1;
        pcwrite = 1;
        inpc = 16'h0000;
```

```
    #100;
```

```
        clkout=0;
        #100
        clkout=1;
        pcwrite=0;
        inpc=16'haaaa;
        #100;
        clkout=0;
        #100;
        clkout=1;
        pcwrite=1;
        inpc=16'hbbbb;
```

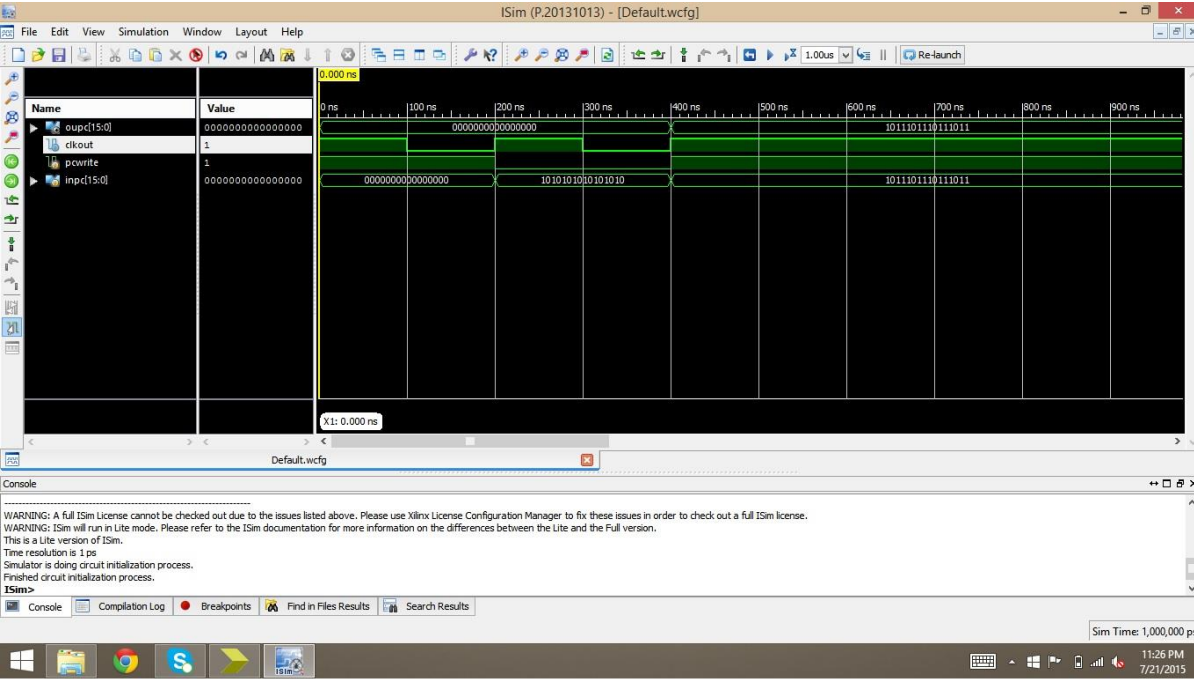
```
        // Wait 100 ns for global reset to finish
        #100;
```

```
        // Add stimulus here
```

```
    end
```

endmodule

Pc Output



MODULE 12

Regfile module test

```

module Regfile(
    input [2:0] ra1,
    input [2:0] ra2,
    input [2:0] wa,
    input [15:0] wd,
    input regwrite,
    input clkout,
    output [15:0] rd1,
    output [15:0] rd2
);
    reg [15:0] Regfile[0:7];
    assign rd1=Regfile[ra1];
    assign rd2=Regfile[ra2];
    always@(posedge clkout)
    begin
        if(regwrite)
        if(wa>0)
        begin
            Regfile[wa]<=wd;
        end
    end
end

initial
//reg [2:0]RegFile[0:3];

begin
$readmemb("reg_data.txt",Regfile);
    $display("r0=%h",Regfile[0]);
    $display("r1==%b",Regfile[1]);
    $display("re==%h and r3==%h",Regfile[2],Regfile[3]);
end

endmodule

```

Regfile_test Code

```

module Regfile_test;

    // Inputs
    reg [2:0] ra1;
    reg [2:0] ra2;
    reg [2:0] wa;

```

```

reg [15:0] wd;
reg regwrite;
reg clkout;

// Outputs
wire [15:0] rd1;
wire [15:0] rd2;

// Instantiate the Unit Under Test (UUT)
Regfile uut (
    .ra1(ra1),
    .ra2(ra2),
    .wa(wa),
    .wd(wd),
    .regwrite(regwrite),
    .clkout(clkout),
    .rd1(rd1),
    .rd2(rd2)
);

initial begin
    // Initialize Inputs
    ra1 = 0;
    ra2 = 1;
    wa = 0;
    wd = 0;
    regwrite = 0;
    clkout = 0;

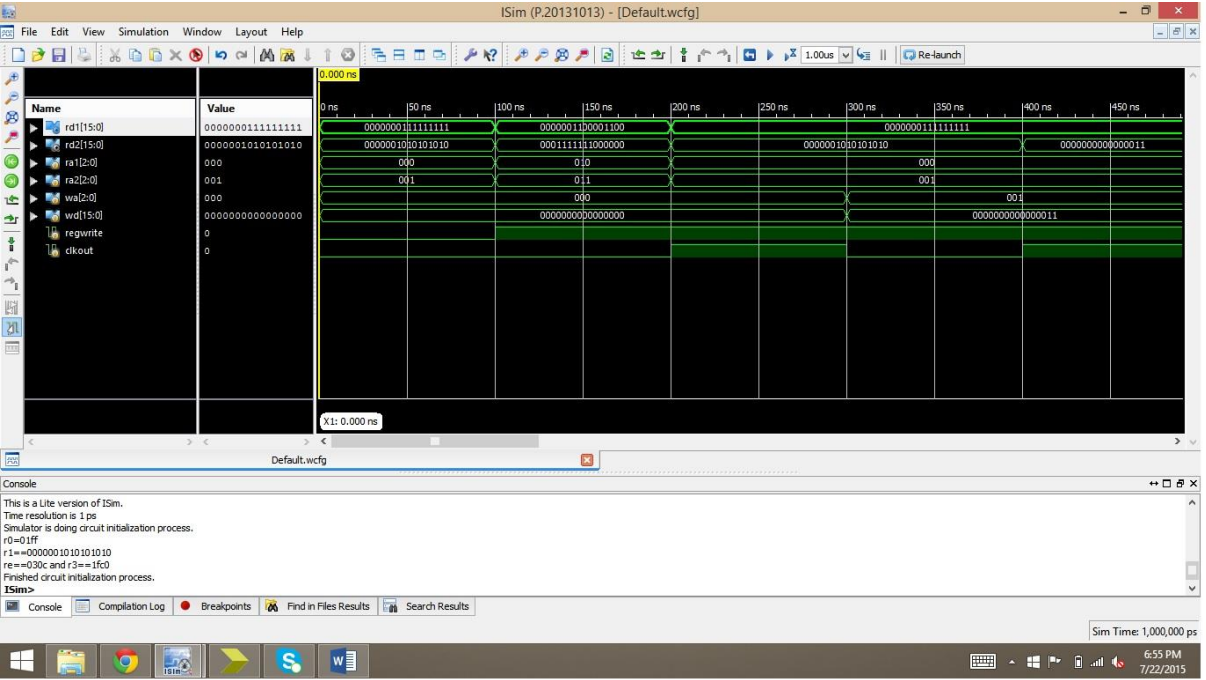
    // Wait 100 ns for global reset to finish
    #100;
    ra1=2;
    ra2=3;
    regwrite=1;
    #100
    ra1=0;ra2=1;
    clkout=1;
    #100
    clkout=0;
    wa=1;
    wd=3;
    #100
    clkout=1;
    #100
    clkout=0;

    // Add stimulus here

end
endmodule

```

Regfile output



MODULE 13

shift reg 16 module code

```

module shift_reg_16(
    input [15:0] oualu,
    input [3:0] shamt,
    input dir,
    output [15:0] oushift
);
    reg [15:0] oushift;

    always @(oualu or shamt or dir)
    begin
        if (dir==0)
        begin
            oushift=oualu << shamt;
        end
        else
            oushift=oualu >> shamt;
        end
    end

endmodule
  
```

shift reg 16 test code

module shift_reg_16_test;

```

// Inputs
reg [15:0] oualu;
reg [3:0] shamt;
reg dir;

// Outputs
wire [15:0] oushift;

// Instantiate the Unit Under Test (UUT)
shift_reg_16 uut (
    .oualu(oualu),
    .shamt(shamt),
    .dir(dir),
    .oushift(oushift)
);

initial begin
  
```

```

// Initialize Inputs
oualu = 0;
shamt = 0;
dir = 0;

// Wait 100 ns for global reset to finish
#100;
oualu = 16'h0011;
shamt = 2;
dir = 1;
#100;
oualu = 16'h0011;
shamt = 2;
dir = 0;

```

```

// Add stimulus here

```

```

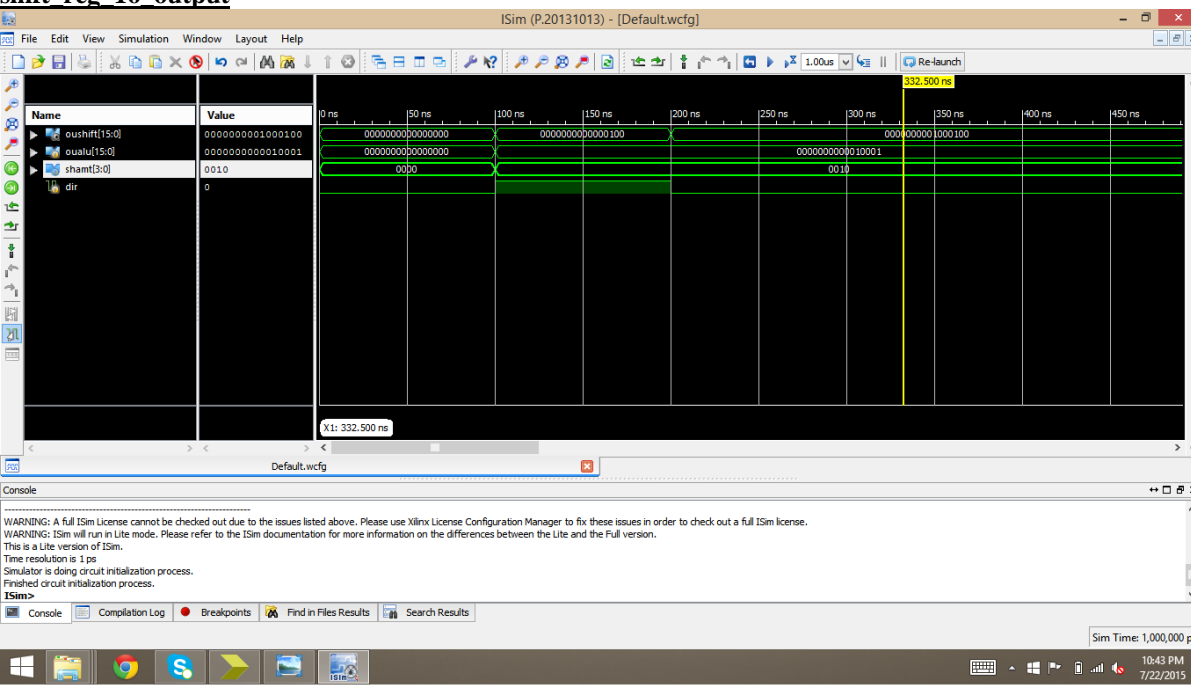
end

```

```

endmodule
shift reg 16 output

```



MODULE 14

Sign extender 5 to 16 module code

```

module sign_extender_5_to_16(
    input [4:0] inse5,
    output [15:0] ouse5
);
    assign ouse5[15:5]={ 11 {inse5[4]}};
    assign ouse5[4:0]=inse5[4:0];
endmodule

```

Sign extender 5 to 16 test code

```

module sign_extender_5_to_16_test;

    // Inputs
    reg [4:0] inse5;

    // Outputs
    wire [15:0] ouse5;

```

```

// Instantiate the Unit Under Test (UUT)
sign_extender_5_to_16 uut (
    .inse5(inse5),
    .ouse5(ouse5)
);

initial begin
    // Initialize Inputs
    inse5 = 0;

    // Wait 100 ns for global reset to finish
    #100;
    inse5=5'b101110;
    #100;
    inse5=5'b011110;

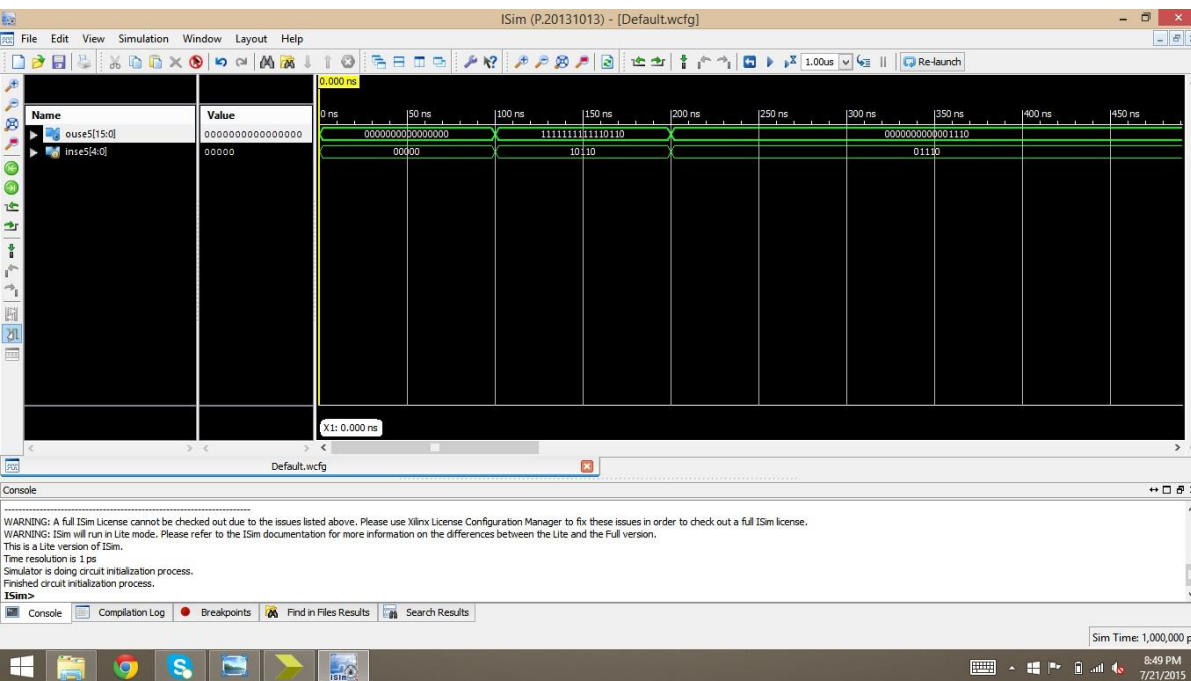
    // Add stimulus here

end

endmodule

```

Sign extender 5 to 16 Output



MODULE 15

sign_extender 8 to 16 module code

```

module sign_extender_8_to_16(
    input [7:0] inse8,
    output [15:0] ouse8
);

    assign ouse8[15:8]={8{inse8[7]}};
    assign ouse8[7:0]=inse8[7:0];

endmodule

```

sign_extender 8 to 16 Test code

```

module sign_extender_8_to_16_test;

    // Inputs
    reg [7:0] inse8;

    // Outputs
    wire [15:0] ouse8;

```

```
// Instantiate the Unit Under Test (UUT)
sign_extender_8_to_16 uut (
    .inse8(inse8),
    .ouse8(ouse8)
);

initial begin
    // Initialize Inputs
    inse8 = 0;

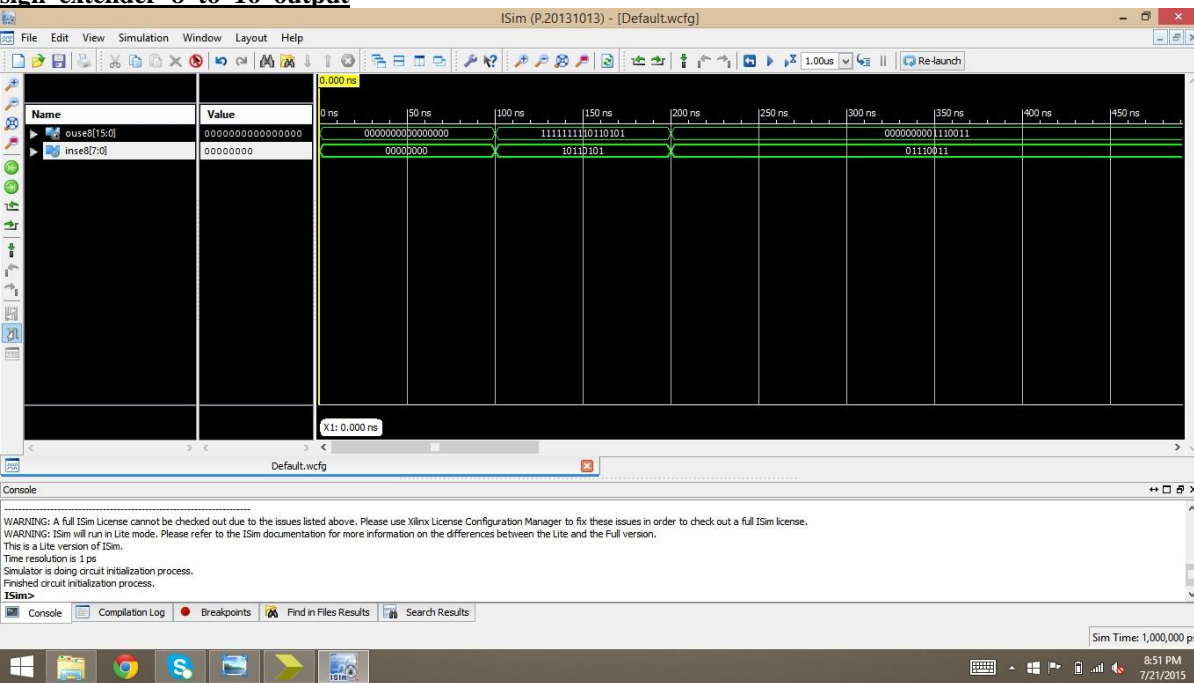
    // Wait 100 ns for global reset to finish
    #100;
    inse8=8'b10110101;
    #100;
    inse8=8'b01110011;

    // Add stimulus here

end
```

endmodule

sign_extender 8 to 16 output



MODULE 16

zero_extender 5 to 16 module code

```
module zero_extender_5_to_16(
    input [4:0] inze5,
    output [15:0] ouze5
);
    assign ouze5[15:0]={ 11'b000000000000,inze5[4:0]};
```

endmodule

zero_extender 5 to 16 test code

```
module zero_extender_5_to_16_test;
```

```
// Inputs
reg [4:0] inze5;
```

```
// Outputs
```

```
wire [15:0] ouze5;
```

```
// Instantiate the Unit Under Test (UUT)
```

```
zero_extender_5_to_16 uut (
    .inze5(inze5),
    .ouze5(ouze5)
);
```

```
initial begin
```

```
    // Initialize Inputs
```

```
    inze5 = 0;
```

```
    // Wait 100 ns for global reset to finish
```

```
    #100;
```

```
    inze5=5'b10110;
```

```
    #100;
```

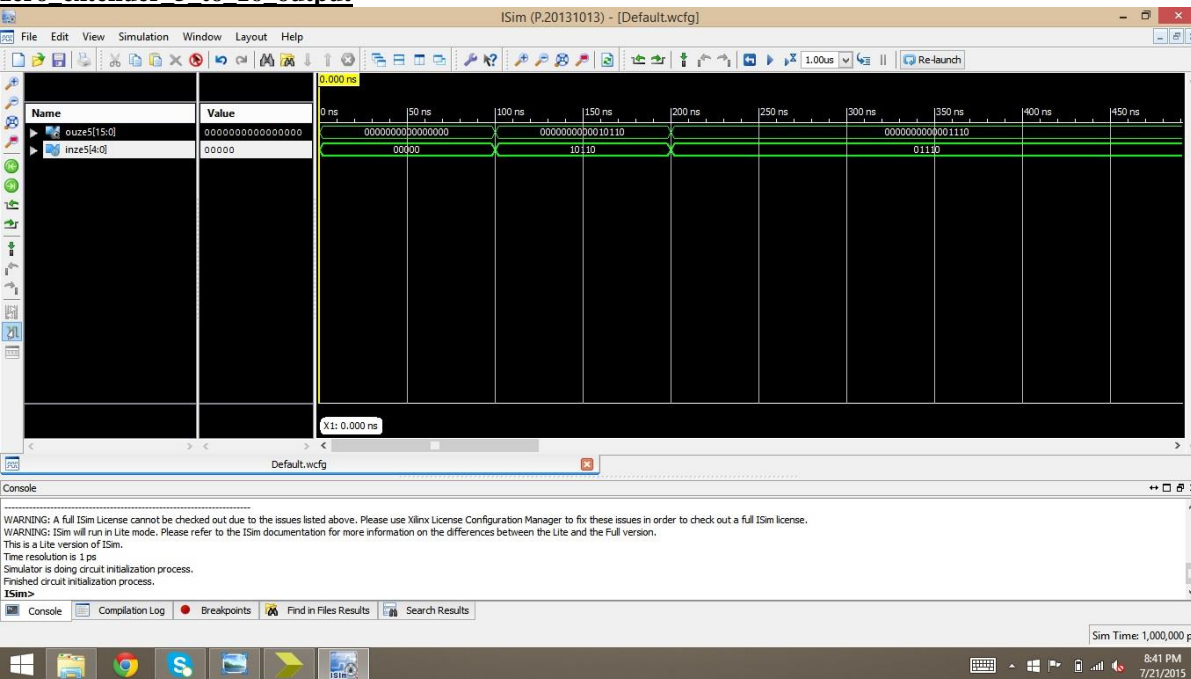
```
    inze5=5'b01110;
```

```
    // Add stimulus here
```

```
end
```

```
endmodule
```

zero_extender 5 to 16 output



MODULE 17

zero_extender 8 to 16 module code

```
module zero_extender_8_to_16(
```

```
    input [7:0] inze8,
```

```
    output [15:0] ouze8
```

```
);
```

```
    assign ouze8[15:0]={8'b000000000000,inze8[7:0]};
```

```
endmodule
```

zero_extender 8 to 16 test code

```
module zero_extender_8_to_16_test;
```

```
    // Inputs
```

```

reg [7:0] inze8;

// Outputs
wire [15:0] ouze8;

// Instantiate the Unit Under Test (UUT)
zero_extender_8_to_16 uut (
    .inze8(inze8),
    .ouze8(ouze8)
);

initial begin
    // Initialize Inputs
    inze8 = 0;

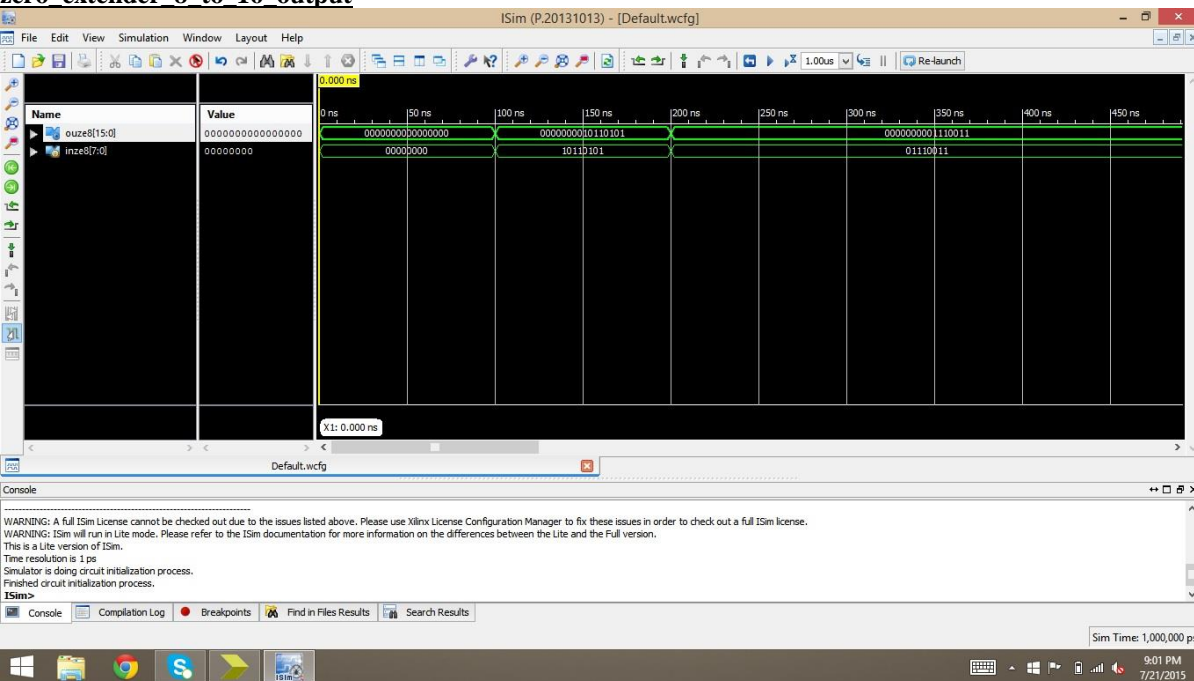
    // Wait 100 ns for global reset to finish
    #100;
    inze8=8'b01110101;
    #100;
    inze8=8'b01110011;
    // Add stimulus here

end

```

endmodule

zero_extender 8 to 16 output



MODULE 18

state register module code

```

module state_register(

input [4:0]nextState,
output [4:0] stateOut
);
clock men(clkout);
reg [4:0] stateReg; //Define Register File
assign stateOut = stateReg; //Continuously reads output

always @(posedge clkout)
begin    stateReg = nextState;

end

```

end


```

initial
begin    stateReg =0;

end

endmodule

state register test code
module state_register_test;

    // Inputs
    reg [4:0] nextState;

    // Outputs
    wire [4:0] stateOut;

    // Instantiate the Unit Under Test (UUT)
    state_register uut (
        .nextState(nextState),
        .stateOut(stateOut)
    );

    initial begin
        // Initialize Inputs
        nextState = 0001;

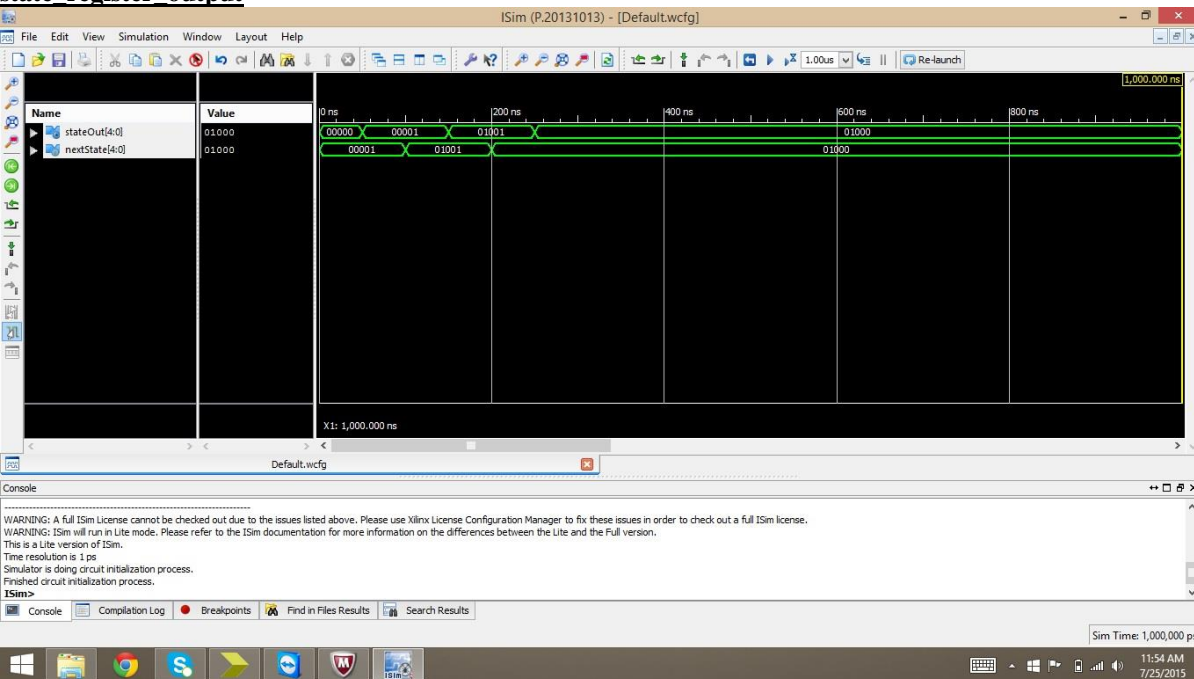
        // Wait 100 ns for global reset to finish
        #100;
        nextState=1001;
        #100;
        nextState=1000;

        // Add stimulus here

    end
endmodule

```

state register output



MODULE 19

Regb module code

```

module regb(
    input [15:0] inregb,
    output [15:0] ouregb
);
reg[15:0]ouregb;

```

```
clock spiderman(clkout);
```

```
always @(posedge clkout)
```

```
begin
```

```
assign ouregb=inregb;
```

```
end
```

```
endmodule
```

Regb test code

```
module regb_test;
```

```
// Inputs
```

```
reg [15:0] inregb;
```

```
// Outputs
```

```
wire [15:0] ouregb;
```

```
// Instantiate the Unit Under Test (UUT)
```

```
regb uut (
```

```
    .inregb(inregb),
```

```
    .ouregb(ouregb)
```

```
);
```

```
initial begin
```

```
// Initialize Inputs
```

```
    inregb = 0;
```

```
// Wait 100 ns for global reset to finish
```

```
#100;
```

```
inregb = 16'b 0001100010110101;
```

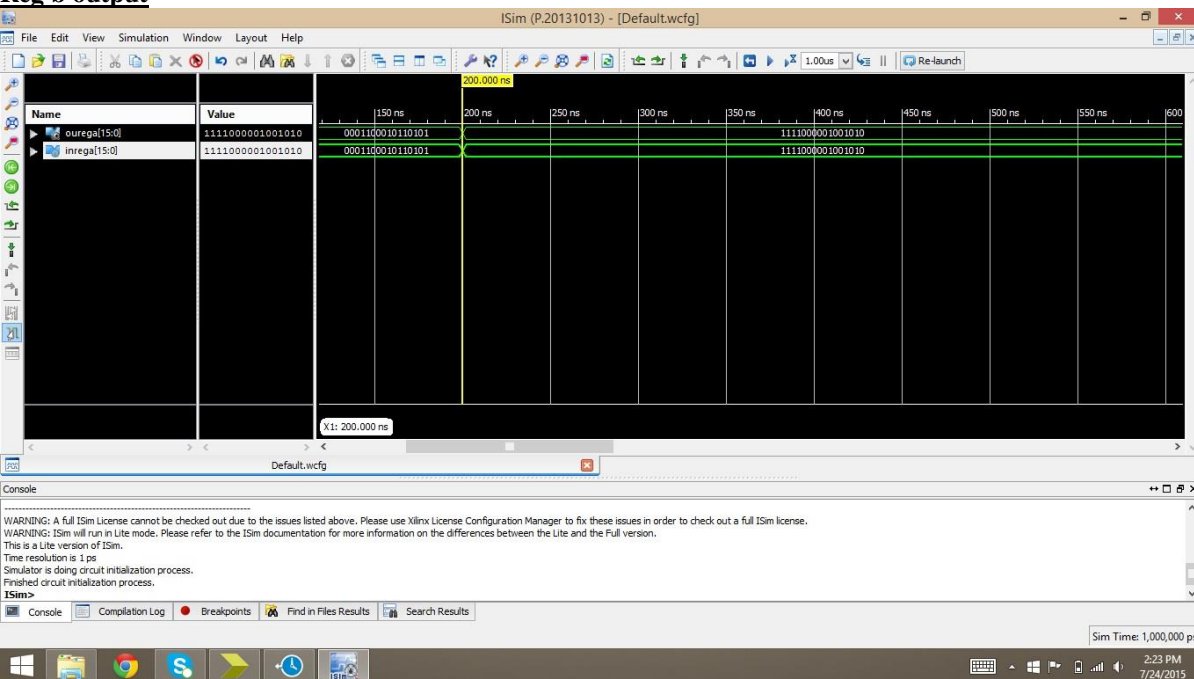
```
#100;
```

```
inregb = 16'b 1111000001001010;
```

```
end
```

```
endmodule
```

Reg b output



Overflow module code

```
module overflow(  
    input a,  
    input b,  
    input c,  
    output reg overflow  
  
);
```

```
always@(a or b or c)  
begin  
    if (a == b)  
        begin  
            if (a == 0)  
                overflow = a + b + c;  
            else  
                overflow = a + b;  
            end  
        else  
            overflow = 0;  
        end  
end
```

```
endmodule
```

Overflow test code

```
module overflow_test1;
```

```
    // Inputs
```

```
    reg a;
```

```
    reg b;
```

```
    reg c;
```

```
    // Outputs
```

```
    wire overflow;
```

```
    // Instantiate the Unit Under Test (UUT)
```

```
    overflow uut (  
        .a(a),  
        .b(b),  
        .c(c),  
        .overflow(overflow)
```

```
    );
```

```
    );
```

```
    );
```

```
    );
```

```
);
```

```
initial begin
```

```
    // Initialize Inputs
```

```
    a = 0;
```

```
    b = 0;
```

```
    c = 0;
```

```
    // Wait 100 ns for global reset to finish
```

```
    #100;
```

```
    a = 1;
```

```
    b = 0;
```

```
    c = 1;
```

```
    #100;
```

```
    a = 0;
```

```
    b = 0;
```

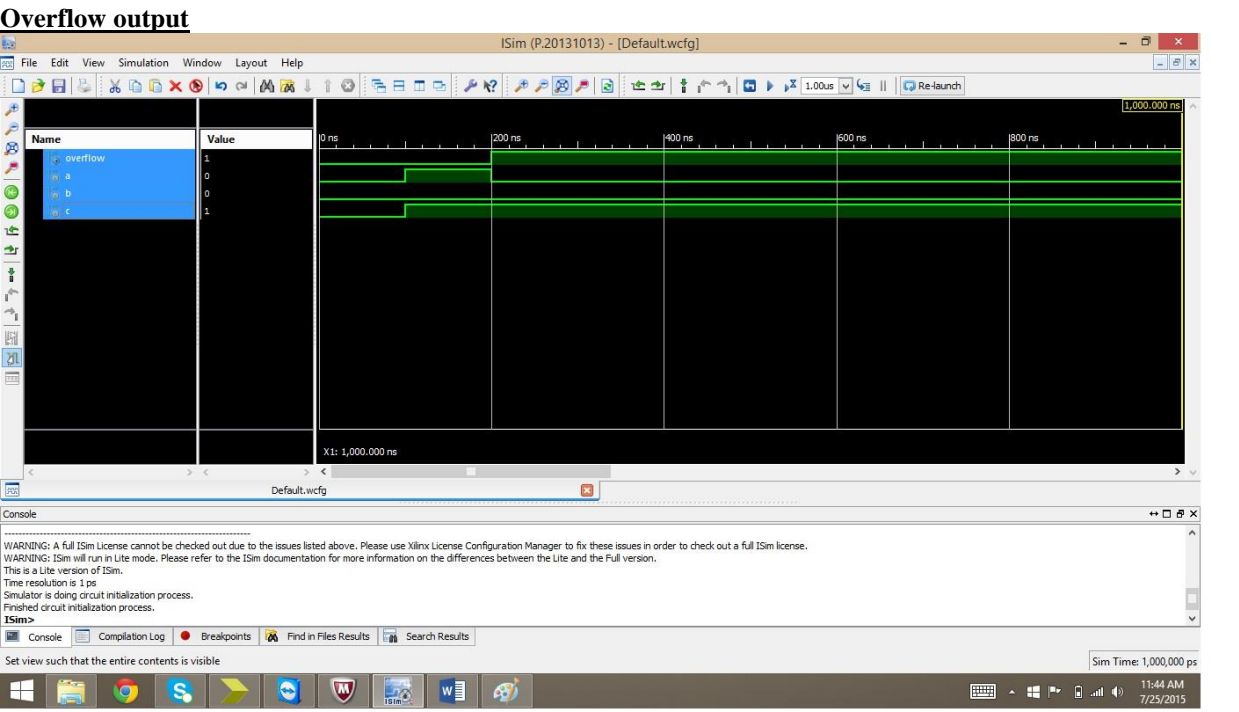
```
    c = 1;
```

```
    #100;
```

```
    // Add stimulus here
```

```
end
```

```
endmodule
```



MODULE 21

Mux8to1 module code

```

module mux_8_to_1(
    input a,
    input b,
    input c,
    input d,
    input e,
    input f,
    input g,
    input h,
    input [2:0] s,
    output y
);
    assign y=s[2]?(s[1]?(s[0]?h:g):(s[0]?f:e)):(s[1]?(s[0]?d:c):(s[0]?b:a));
  
```

endmodule

Mux8to1 test code

```

module mux_8_to_1_test;
  
```

// Inputs

```

reg a;
reg b;
reg c;
reg d;
reg e;
reg f;
reg g;
reg h;
reg [2:0] s;
  
```

// Outputs

```

wire y;
  
```

// Instantiate the Unit Under Test (UUT)

```

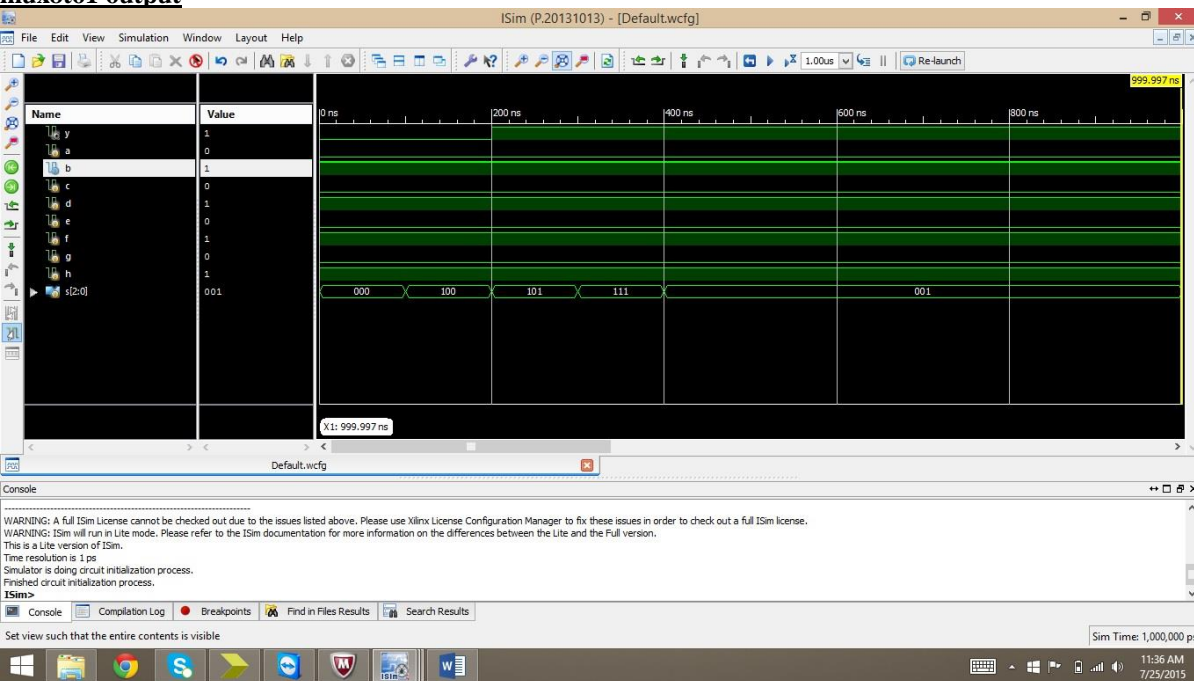
mux_8_to_1 uut (
    .a(a),
    .b(b),
    .c(c),
    .d(d),
    .e(e),
  
```

```
initial begin
    // Initialize Inputs
```

```
// Wait 100 ns for global reset to finish
#100;
s=3'b100;
#100;
s=3'b101;
#100;
s=3'b111;
#100;
s=3'b001;
```

end

mux8to1 output



MODULE 21

mux 2 to 1 module code

```
module mux_2_to_1_I_or_D(
    input iniod1,
    input iniod2,
    input iniods,
    output ouiod

```

```
);
assign ouiod =(iniods==0)?iniod1:iniod2;
endmodule
```

mux 2 to 1 test code

```
module mux_2_to_1_I_or_D_test;
```

```
// Inputs
reg iniod1;
reg iniod2;
reg iniods;
```

```
// Outputs
wire ouiod;
```

```
// Instantiate the Unit Under Test (UUT)
```

```
mux_2_to_1_I_or_D uut (
    .iniod1(iniod1),
    .iniod2(iniod2),
    .iniods(iniods),
    .ouiod(ouiod)

```

```
);
```

```
initial begin
```

```
// Initialize Inputs
```

```
iniod1 = 0;
iniod2 = 0;
iniods = 0;
```

```
// Wait 100 ns for global reset to finish
```

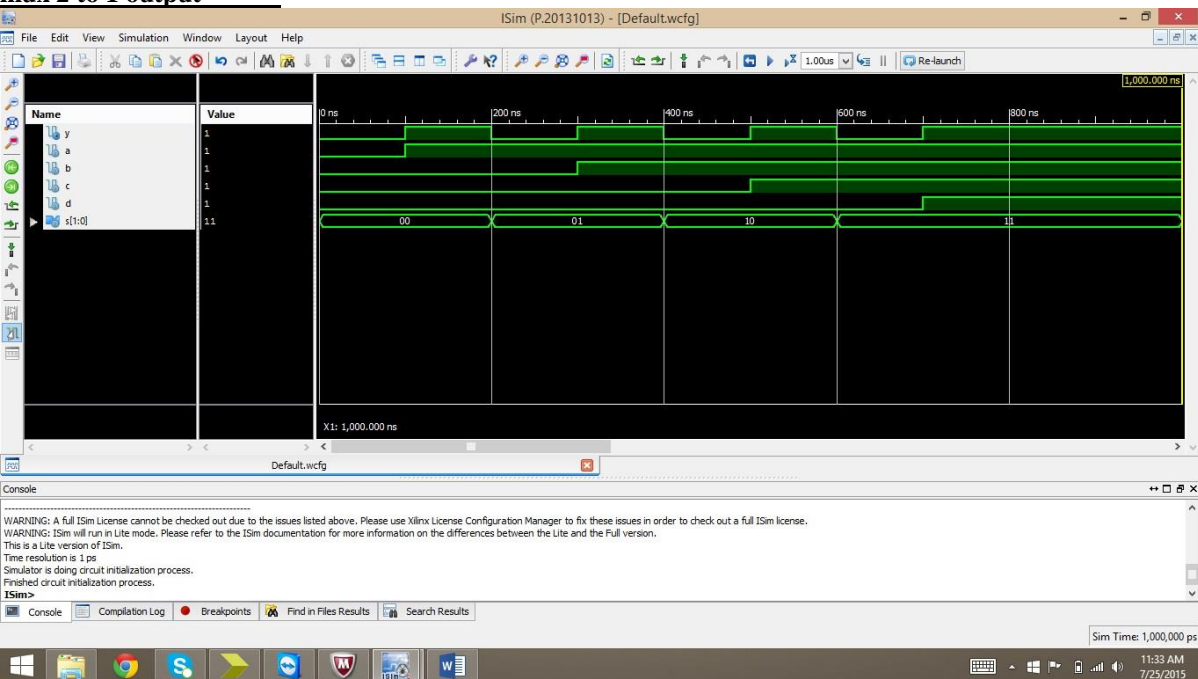
```
#100;
iniod1 = 1;
#100;
iniods = 1;
#100;
iniod2 = 1;
```

```
// Add stimulus here
```

```
end
```

```
endmodule
```

mux 2 to 1 output



MODULE 22

reg_aluout module code

```
module reg_aluout(  
    input [15:0] inaluout,  
    output [15:0] oualuout  
);  
    reg[15:0]oualuout;  
  
    clock almansor(clkout);
```

```
always @(posedge clkout)  
begin  
    assign oualuout=inaluout;
```

```
end  
endmodule
```

reg_aluout test code

```
module reg_aluout_test;
```

```
    // Inputs  
    reg [15:0] inaluout;
```

```
    // Outputs  
    wire [15:0] oualuout;
```

```
    // Instantiate the Unit Under Test (UUT)
```

```
    reg_aluout uut (  
        .inaluout(inaluout),  
        .oualuout(oualuout)  
    );
```

```
    initial begin
```

```
        // Initialize Inputs
```

```
        inaluout = 0;
```

```
        // Wait 100 ns for global reset to finish
```

```
#100;
```

```
inaluout = 16'b 0001100010110101;
```

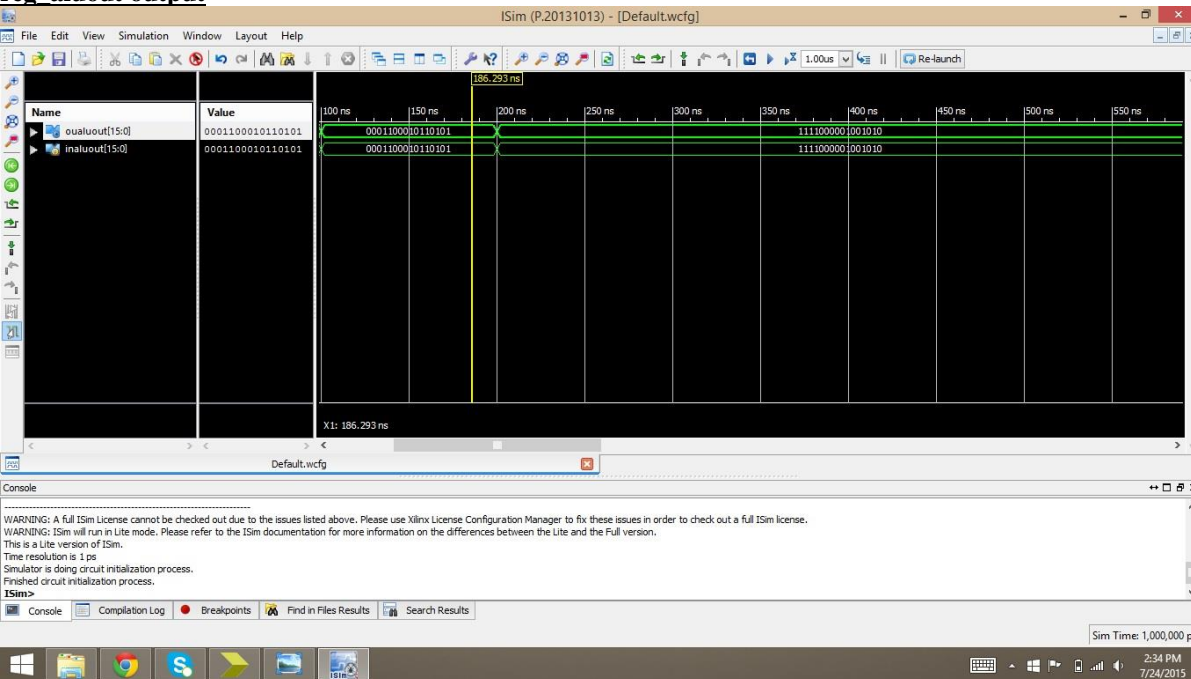
```
#100;
```

```
inaluout = 16'b 1111000001001010;
```

```
    end
```

```
endmodule
```

reg_aluout output



MODULE 23

Reg a module code

```
module rega(  
    input [15:0] inrega,  
    output [15:0] ourega  
);  
  
reg[15:0]ourega;  
  
clock rosemaid(clkout);  
  
always @(posedge clkout)  
begin  
    assign ourega=inrega;
```

```
end  
endmodule
```

Reg a test code

```
module rega_test;  
  
    // Inputs  
    reg [15:0] inrega;  
  
    // Outputs  
    wire [15:0] ourega;  
  
    // Instantiate the Unit Under Test (UUT)  
    rega uut (  
        .inrega(inrega),  
        .ourega(ourega)  
    );  
  
    initial begin  
        // Initialize Inputs  
        inrega = 0;  
        // Wait 100 ns for global reset to finish  
  
#100;  
        inrega = 16'b 0001100010110101;  
#100;  
        inrega = 16'b 1111000001001010;  
    end
```

```
endmodule
```

Reg a output

The screenshot displays the ISim software interface, which is used for digital logic simulation. The top panel shows a truth table for a 2-to-1 multiplexer. The inputs are 'eurega[15:0]' and 'inrega[15:0]', and the output is '1111000001001010'. The middle panel shows a timing diagram with a clock signal 'X1: 200.000 ns' and a data signal '1111000001001010'. The bottom panel shows the console output with a warning message about the ISim license and a message indicating the simulation is finished.

Name	Value
eurega[15:0]	1111000001001010
inrega[15:0]	1111000001001010

Timing Diagram:

X1: 200.000 ns

1111000001001010

Console:

WARNING: A full ISim License cannot be checked out due to the issues listed above. Please use Xilinx License Configuration Manager to fix these issues in order to check out a full ISim license.
 WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
 This is a Lite version of ISim.
 Time resolution is 1 ps
 Simulator is doing circuit initialization process.
 Finished circuit initialization process.
 ISim>

Sim Time: 1,000,000 ps

