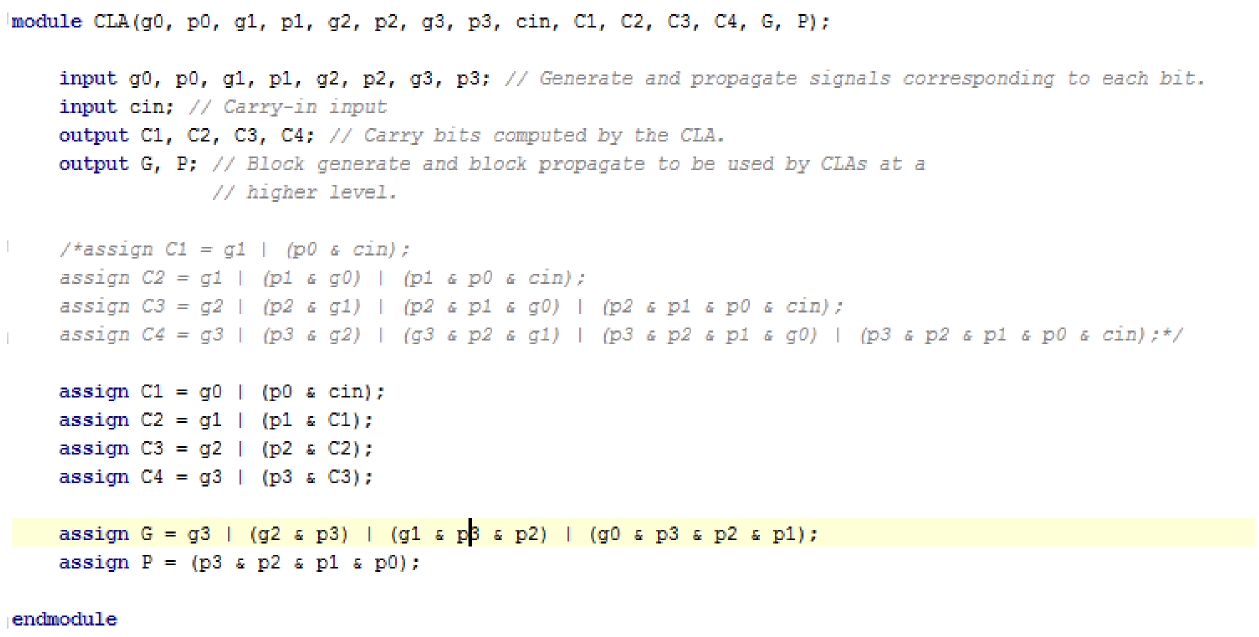
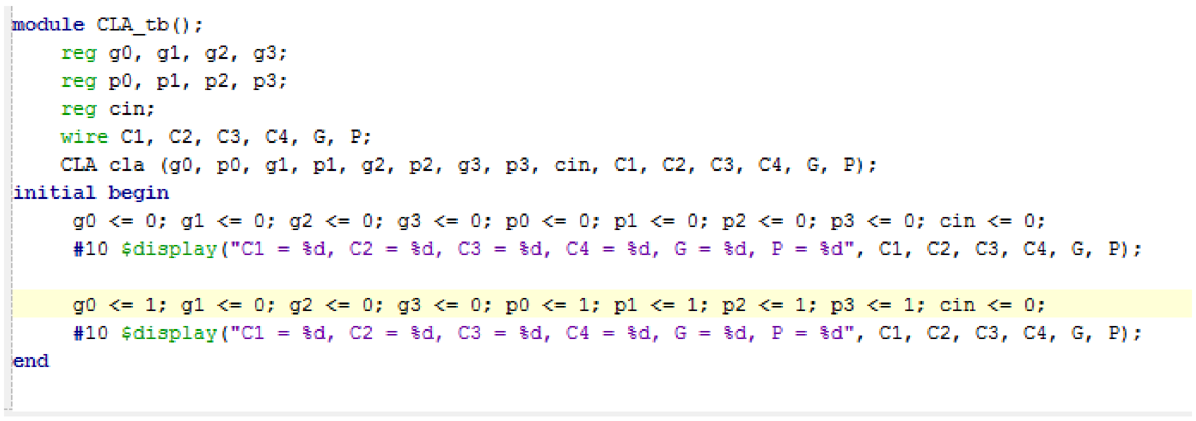


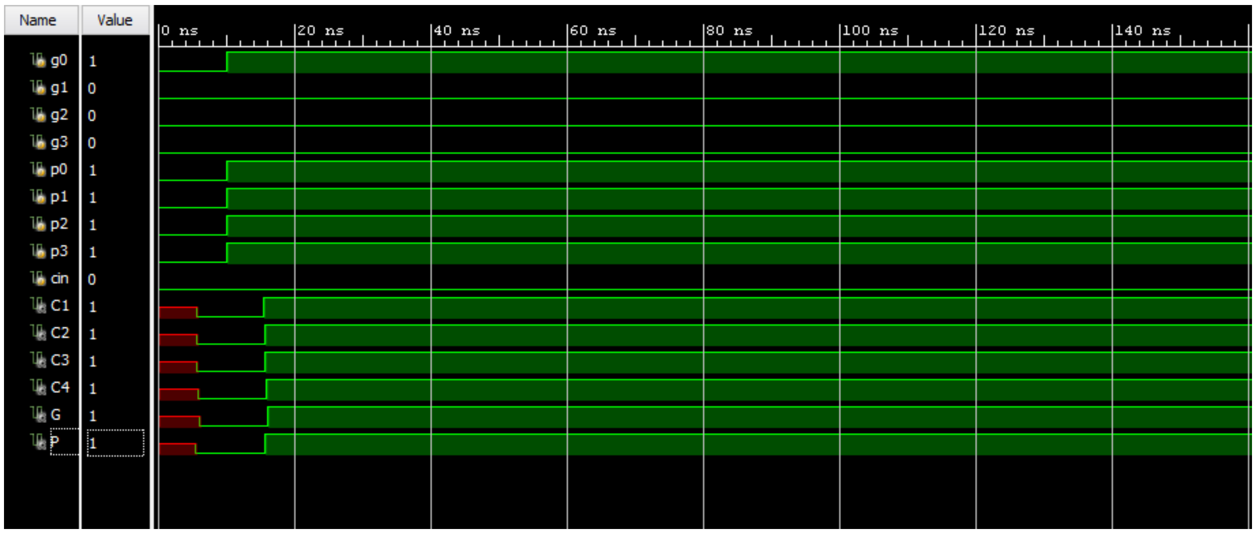
```
module CLA(g0, p0, g1, p1, g2, p2, g3, p3, cin, C1, C2, C3, C4, G, P);

    input g0, p0, g1, p1, g2, p2, g3, p3; // Generate and propagate signals corresponding to each bit.
    input cin; // Carry-in input
    output C1, C2, C3, C4; // Carry bits computed by the CLA.
    output G, P; // Block generate and block propagate to be used by CLAs at a
                 // higher level.

    /*assign C1 = g1 | (p0 & cin);
    assign C2 = g1 | (p1 & g0) | (p1 & p0 & cin);
    assign C3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & cin);
    assign C4 = g3 | (p3 & g2) | (g3 & p2 & g1) | (p3 & p2 & p1 & g0) | (p3 & p2 & p1 & p0 & cin);*/

    assign C1 = g0 | (p0 & cin);
    assign C2 = g1 | (p1 & C1);
    assign C3 = g2 | (p2 & C2);
    assign C4 = g3 | (p3 & C3);

    assign G = g3 | (g2 & p3) | (g1 & p3 & p2) | (g0 & p3 & p2 & p1);
    assign P = (p3 & p2 & p1 & p0);

endmodule
```

Test Bench

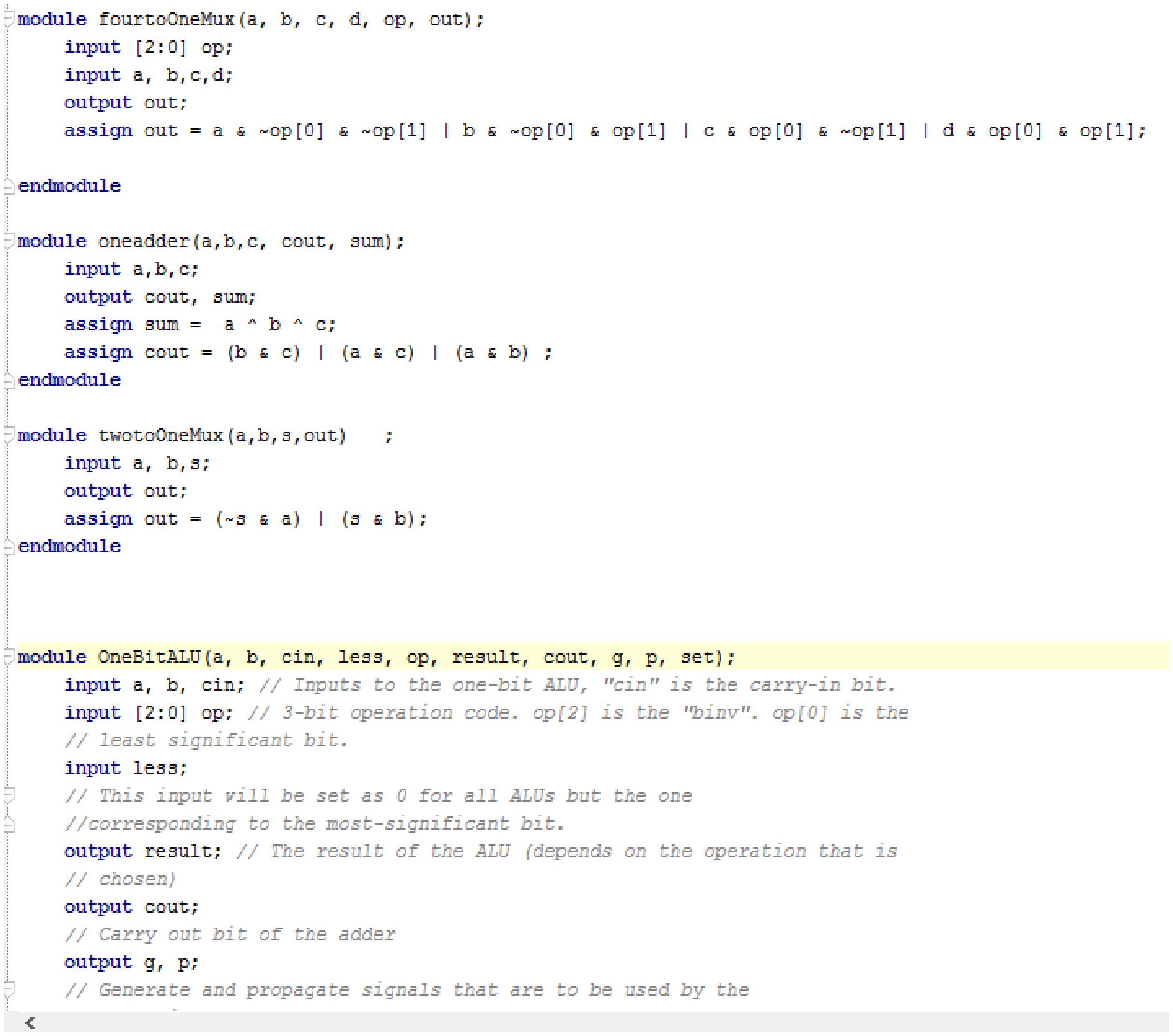


```
module CLA_tb();
    reg g0, g1, g2, g3;
    reg p0, p1, p2, p3;
    reg cin;
    wire C1, C2, C3, C4, G, P;
    CLA cla (g0, p0, g1, p1, g2, p2, g3, p3, cin, C1, C2, C3, C4, G, P);
initial begin
     g0 <= 0; g1 <= 0; g2 <= 0; g3 <= 0; p0 <= 0; p1 <= 0; p2 <= 0; p3 <= 0; cin <= 0;
     #10 $display("C1 = %d, C2 = %d, C3 = %d, C4 = %d, G = %d, P = %d", C1, C2, C3, C4, G, P);

     g0 <= 1; g1 <= 0; g2 <= 0; g3 <= 0; p0 <= 1; p1 <= 1; p2 <= 1; p3 <= 1; cin <= 0;
     #10 $display("C1 = %d, C2 = %d, C3 = %d, C4 = %d, G = %d, P = %d", C1, C2, C3, C4, G, P);
end
```

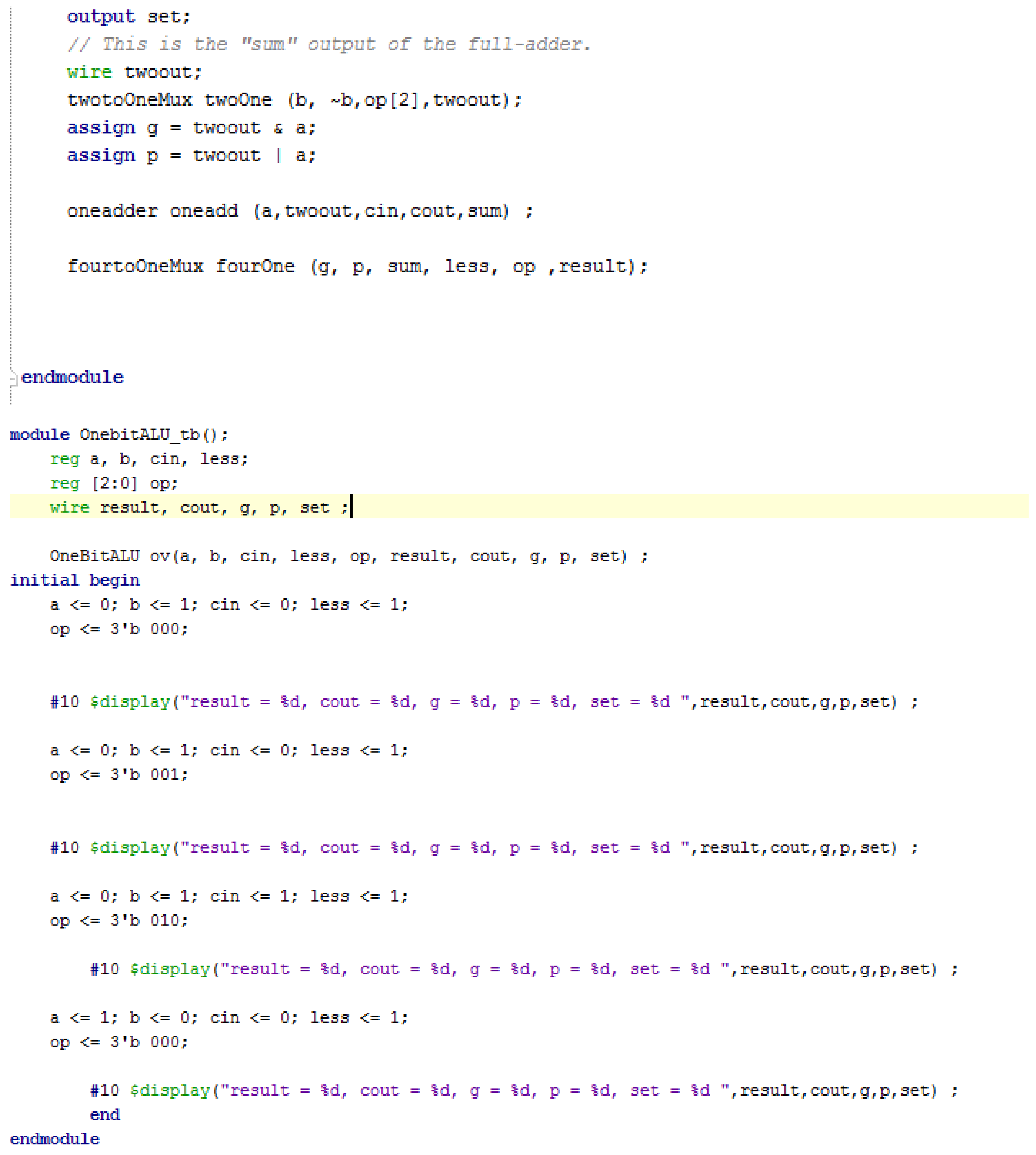
Waveform



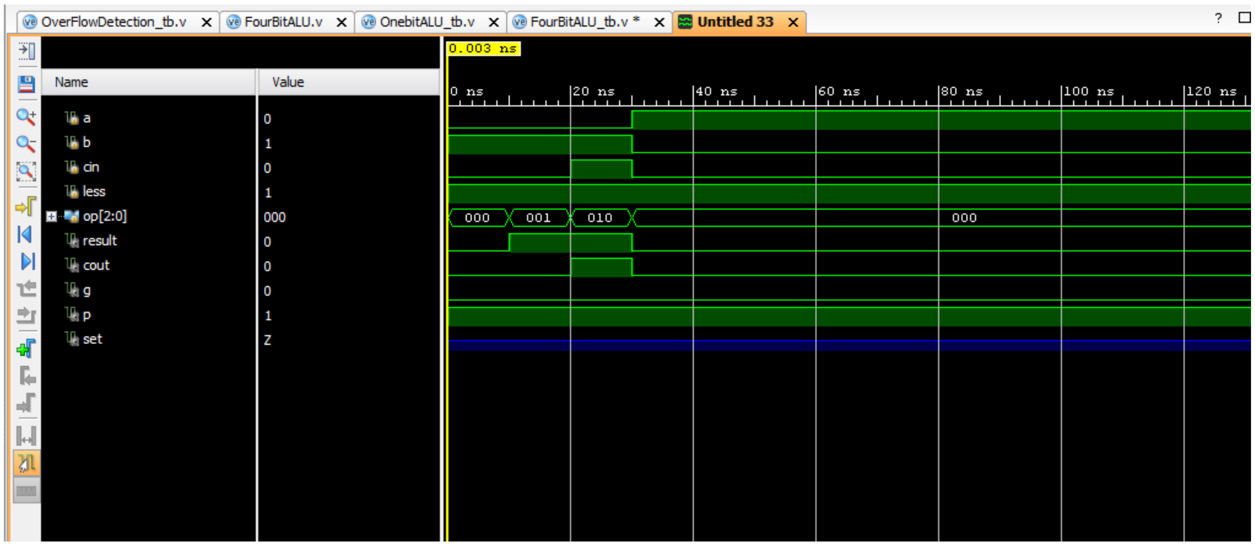
Name	Value
g0	1
g1	0
g2	0
g3	0
p0	1
p1	1
p2	1
p3	1
cin	0
C1	1
C2	1
C3	1
C4	1
G	1
P	1



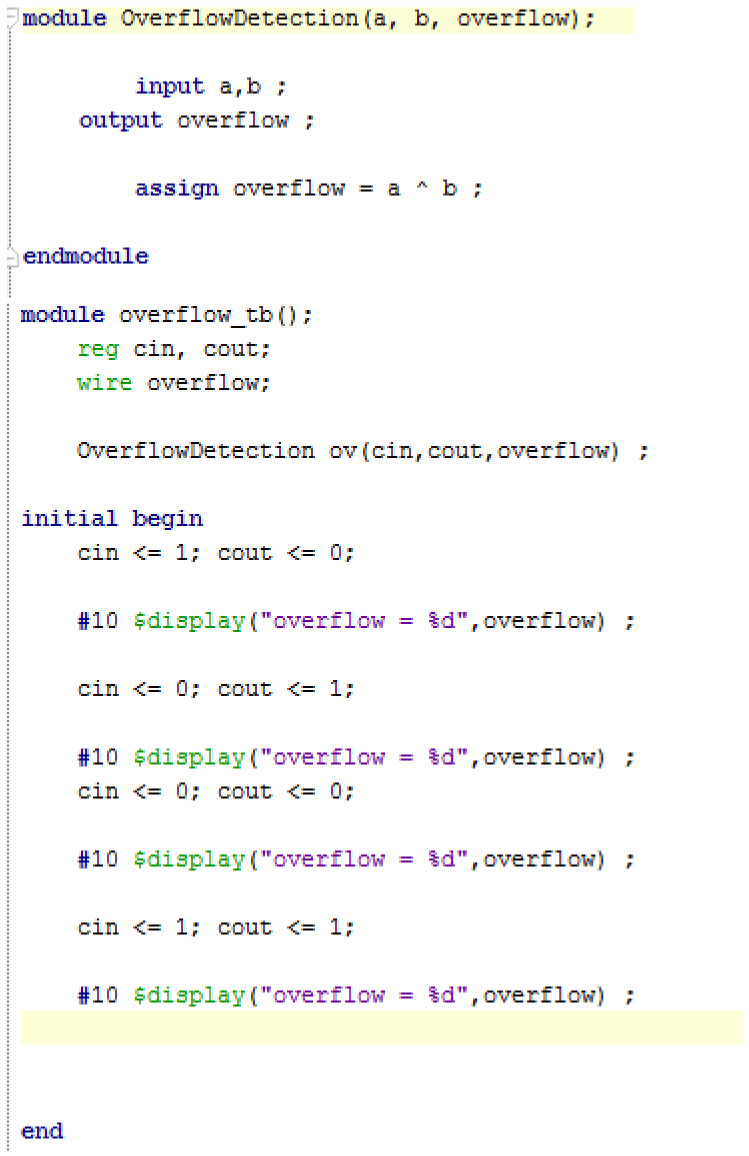
```
module fourtoOneMux(a, b, c, d, op, out);
    input [2:0] op;
    input a, b,c,d;
    output out;
    assign out = a & ~op[0] & ~op[1] | b & ~op[0] & op[1] | c & op[0] & ~op[1] | d & op[0] & op[1];

endmodule

module oneadder(a,b,c, cout, sum);
    input a,b,c;
    output cout, sum;
    assign sum =  a ^ b ^ c;
    assign cout = (b & c) | (a & c) | (a & b) ;
endmodule

module twotoOneMux(a,b,s,out)   ;
    input a, b,s;
    output out;
    assign out = (~s & a) | (s & b);
endmodule



module OneBitALU(a, b, cin, less, op, result, cout, g, p, set);
    input a, b, cin; // Inputs to the one-bit ALU, "cin" is the carry-in bit.
    input [2:0] op; // 3-bit operation code. op[2] is the "binv". op[0] is the
    // least significant bit.
    input less;
    // This input will be set as 0 for all ALUs but the one
    //corresponding to the most-significant bit.
    output result; // The result of the ALU (depends on the operation that is
    // chosen)
    output cout;
    // Carry out bit of the adder
    output g, p;
    // Generate and propagate signals that are to be used by the
```



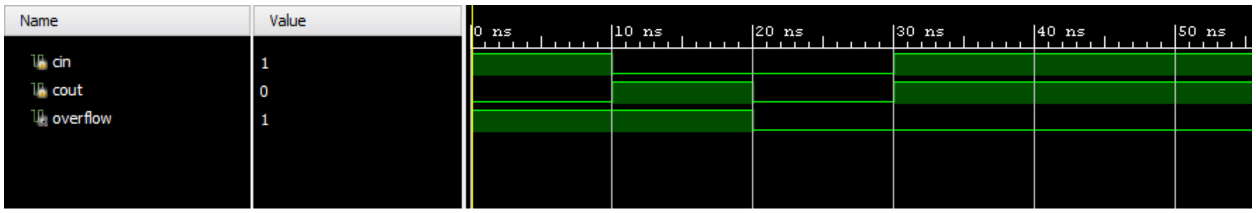
```
     output set;
     // This is the "sum" output of the full-adder.
     wire twoout;
     twotoOneMux twoOne (b, ~b,op[2],twoout);
     assign g = twoout & a;
     assign p = twoout | a;

     oneadder oneadd (a,twoout,cin,cout,sum) ;

     fourtoOneMux fourOne (g, p, sum, less, op ,result);



endmodule

module OnebitALU_tb();
    reg a, b, cin, less;
    reg [2:0] op;
    wire result, cout, g, p, set ;

    OneBitALU ov(a, b, cin, less, op, result, cout, g, p, set) ;
initial begin
    a <= 0; b <= 1; cin <= 0; less <= 1;
    op <= 3'b 000;


    #10 $display("result = %d, cout = %d, g = %d, p = %d, set = %d ",result,cout,g,p,set) ;

    a <= 0; b <= 1; cin <= 0; less <= 1;
    op <= 3'b 001;


    #10 $display("result = %d, cout = %d, g = %d, p = %d, set = %d ",result,cout,g,p,set) ;

    a <= 0; b <= 1; cin <= 1; less <= 1;
    op <= 3'b 010;

        #10 $display("result = %d, cout = %d, g = %d, p = %d, set = %d ",result,cout,g,p,set) ;

    a <= 1; b <= 0; cin <= 0; less <= 1;
    op <= 3'b 000;

        #10 $display("result = %d, cout = %d, g = %d, p = %d, set = %d ",result,cout,g,p,set) ;
        end
endmodule
```



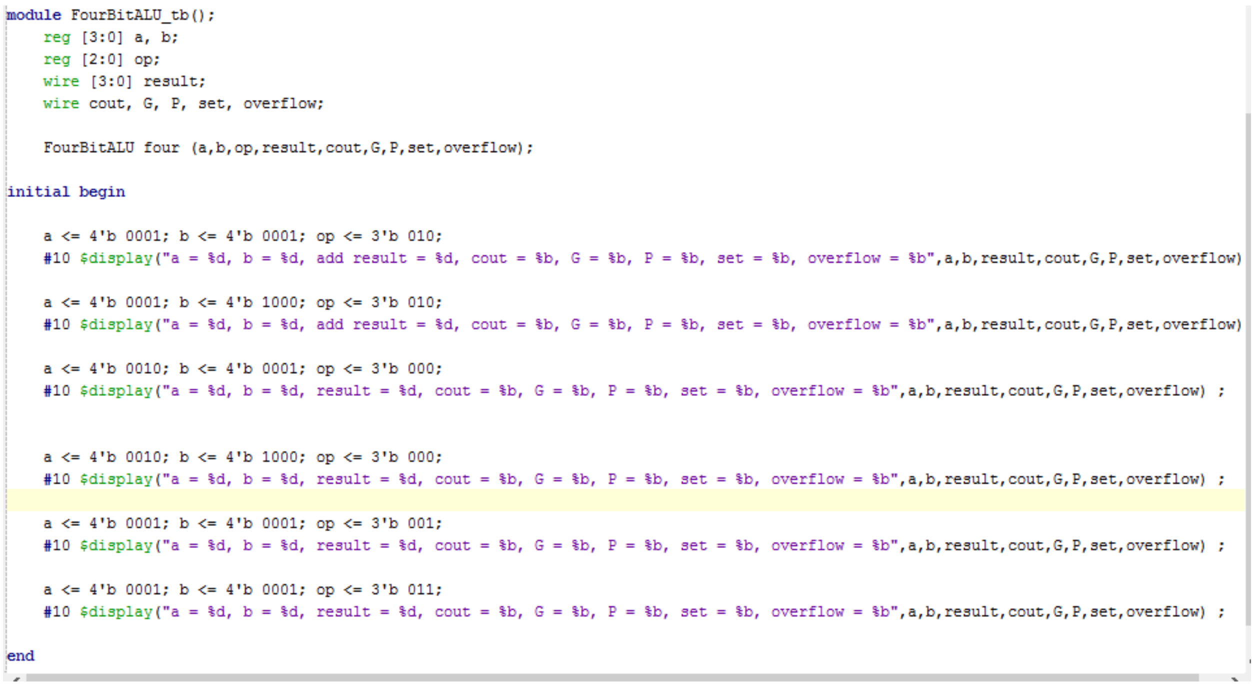
Name	Value
a	0
b	1
cin	0
less	1
op[2:0]	000
result	0
cout	0
g	0
p	1
set	Z



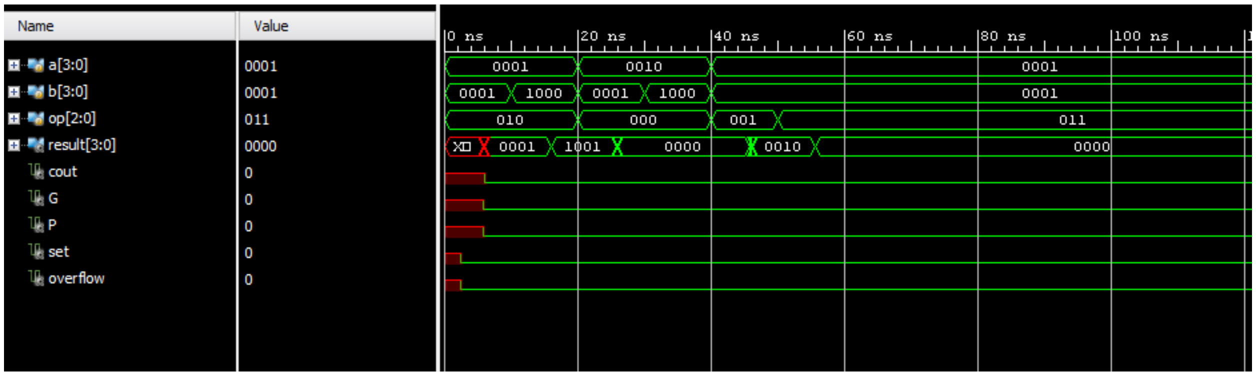
```verilog
module OverflowDetection(a, b, overflow);

        input a,b ;
    output overflow ;

        assign overflow = a ^ b ;

endmodule

module overflow_tb();
    reg cin, cout;
    wire overflow;

    OverflowDetection ov(cin,cout,overflow) ;

initial begin
    cin <= 1; cout <= 0;

    #10 $display("overflow = %d",overflow) ;

    cin <= 0; cout <= 1;

    #10 $display("overflow = %d",overflow) ;
    cin <= 0; cout <= 0;

    #10 $display("overflow = %d",overflow) ;

    cin <= 1; cout <= 1;

    #10 $display("overflow = %d",overflow) ;



end
```



```
module FourBitALU(a, b, op, result, cout, G, P, set, overflow);
    input [3:0] a, b; // Inputs to the one-bit ALU.
    input [2:0] op; // 3-bit operation code. op[2] is the "binv". op[0] is the
    // least significant bit.
    output [3:0] result; // The result of the ALU (depends on the operation that
    // is chosen)
    output cout;
    // Carry-out bit of the ALU
    output G, P;
    // Block generate and propagate of the four-bit ALU
    output set;
    // This is the set output of the most significant ALU block
    output overflow; // This bit indicates that an overflow has occurred.
    // (Ignores what operation is chosen for ALU)
//  wire C1, C2, C3, g0,g1,g2,g3,p0,p1,p2,p3 ;
//  wire r1,r2,r3,r4              ;
    wire C1,C2,C3,C4;
    wire[3:0] C;
    wire[3:0] g,p;
    wire notused;
    wire cin    ;
    wire [3:0]  temps;
    assign cin = op[2]  ;

    CLA carLA(g[0], p[0], g[1], p[1], g[2], p[2], g[3], p[3], op[2], C1, C2, C3, cout, G, P);

    OneBitALU alu0(a[0], b[0], cin, temps[3], op, result[0], C[0], g[0], p[0], temps[0]);
    OneBitALU alu1(a[1], b[1], C1, 1'b0, op, result[1], C[1], g[1], p[1], temps[1]);
    OneBitALU alu2(a[2], b[2], C2, 1'b0, op, result[2], C[2], g[2], p[2], temps[2]);
    OneBitALU alu3(a[3], b[3], C3, 1'b0, op, result[3], C[3], g[3], p[3], temps[3]);

    assign set = temps[3]                ;
    OverflowDetection over(C[3], cout, overflow);
```



```
module FourBitALU_tb();
    reg [3:0] a, b;
    reg [2:0] op;
    wire [3:0] result;
    wire cout, G, P, set, overflow;

    FourBitALU four (a,b,op,result,cout,G,P,set,overflow);

initial begin

    a <= 4'b 0001; b <= 4'b 0001; op <= 3'b 010;
    #10 $display("a = %d, b = %d, add result = %d, cout = %b, G = %b, P = %b, set = %b, overflow = %b",a,b,result,cout,G,P,set,overflow)

    a <= 4'b 0001; b <= 4'b 1000; op <= 3'b 010;
    #10 $display("a = %d, b = %d, add result = %d, cout = %b, G = %b, P = %b, set = %b, overflow = %b",a,b,result,cout,G,P,set,overflow)

    a <= 4'b 0010; b <= 4'b 0001; op <= 3'b 000;
    #10 $display("a = %d, b = %d, result = %d, cout = %b, G = %b, P = %b, set = %b, overflow = %b",a,b,result,cout,G,P,set,overflow) ;


    a <= 4'b 0010; b <= 4'b 1000; op <= 3'b 000;
    #10 $display("a = %d, b = %d, result = %d, cout = %b, G = %b, P = %b, set = %b, overflow = %b",a,b,result,cout,G,P,set,overflow) ;

    a <= 4'b 0001; b <= 4'b 0001; op <= 3'b 001;
    #10 $display("a = %d, b = %d, result = %d, cout = %b, G = %b, P = %b, set = %b, overflow = %b",a,b,result,cout,G,P,set,overflow) ;

    a <= 4'b 0001; b <= 4'b 0001; op <= 3'b 011;
    #10 $display("a = %d, b = %d, result = %d, cout = %b, G = %b, P = %b, set = %b, overflow = %b",a,b,result,cout,G,P,set,overflow) ;

end
```



Name	Value
a[3:0]	0001
b[3:0]	0001
op[2:0]	011
result[3:0]	0000
cout	0
G	0
P	0
set	0
overflow	0