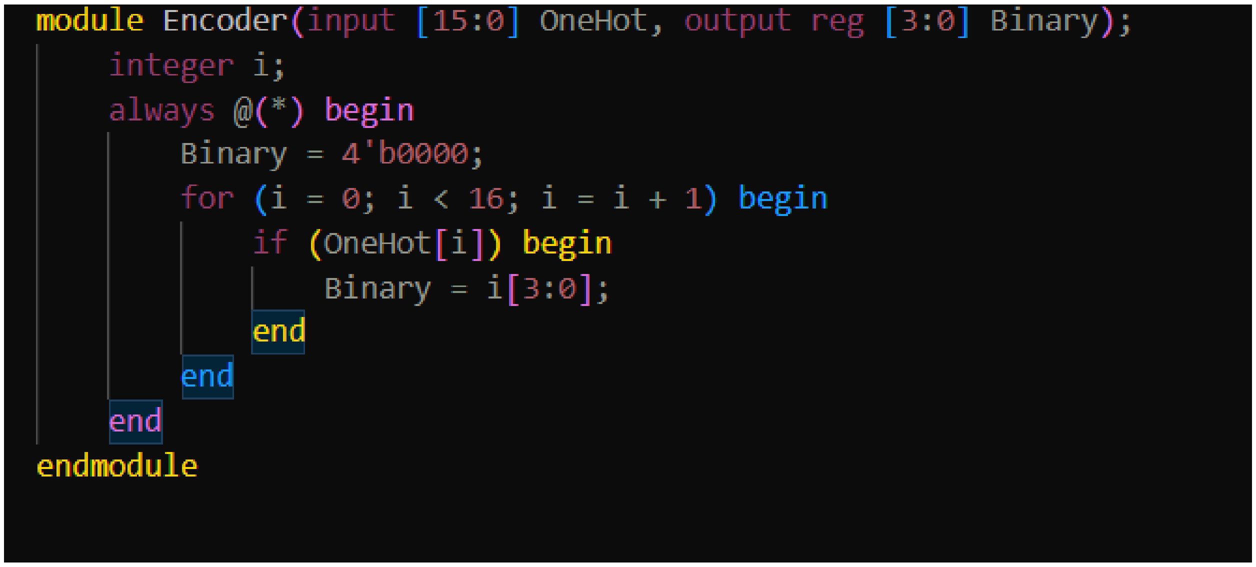
CA2

Sadra Madayeni Avval

810102564

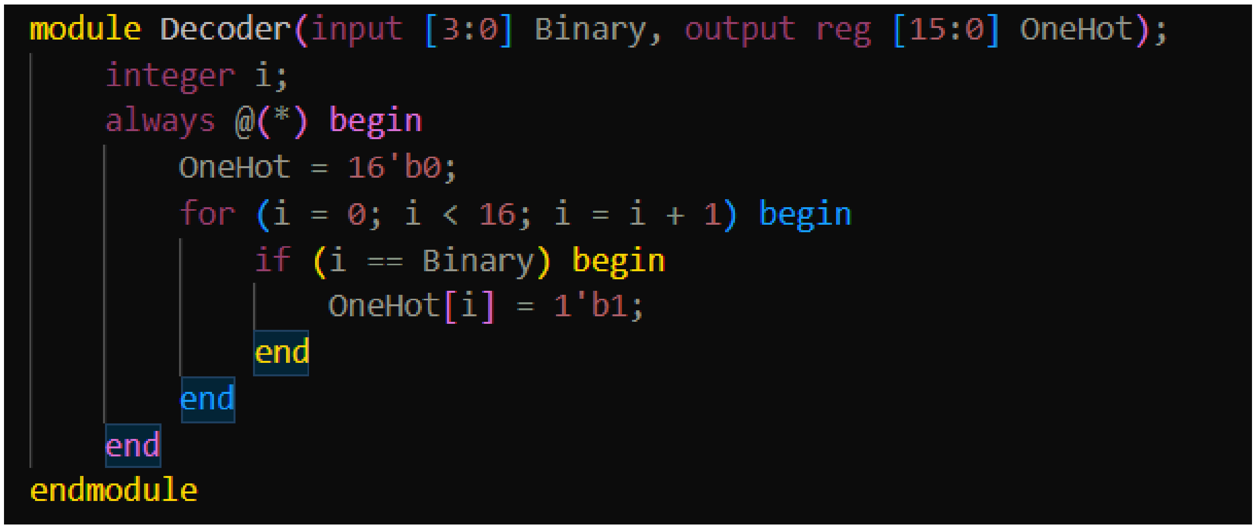
First Of All We Need To Implement What We Need In This Computer-Assignment.

First Of All We Have A OneHot Number That We Should Convert It To a Binary Number
So We Can Implement It Like This



```
module Encoder(input [15:0] OneHot, output reg [3:0] Binary);
    integer i;
    always @(*) begin
        Binary = 4'b0000;
        for (i = 0; i < 16; i = i + 1) begin
            if (OneHot[i]) begin
                Binary = i[3:0];
            end
        end
    end
endmodule
```

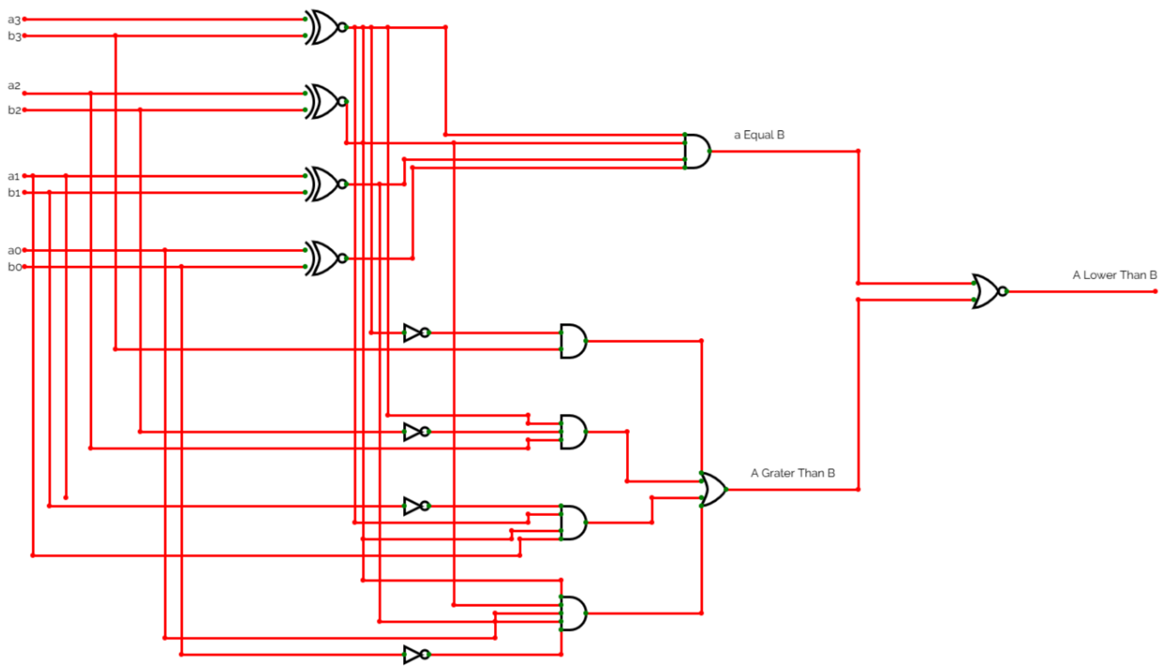
After That We Need A Decoder To Convert Our Binary Input To OneHot Number



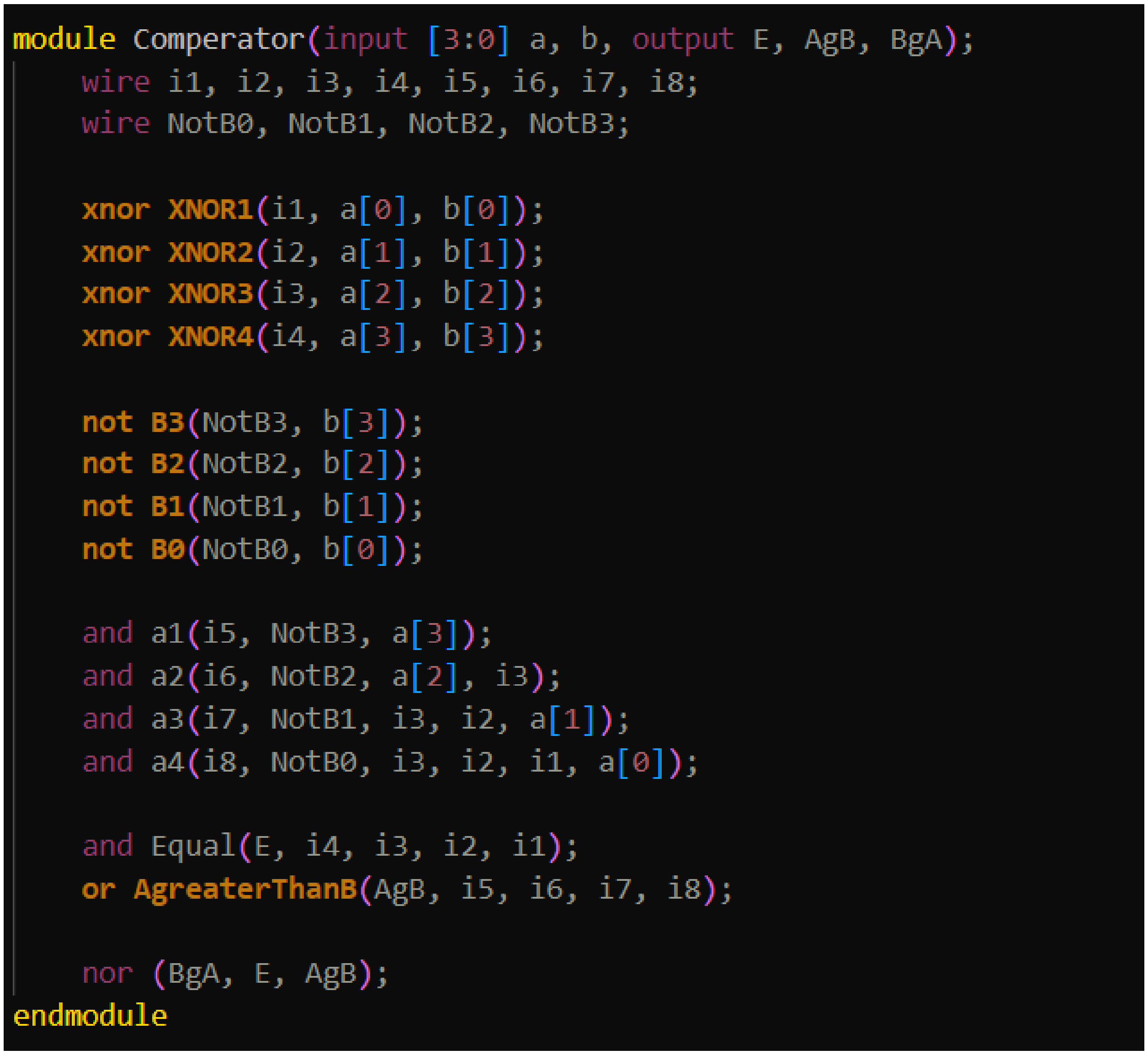
```verilog
module Decoder(input [3:0] Binary, output reg [15:0] OneHot);
    integer i;
    always @(*) begin
        OneHot = 16'b0;
        for (i = 0; i < 16; i = i + 1) begin
            if (i == Binary) begin
                OneHot[i] = 1'b1;
            end
        end
    end
endmodule
```

After That We Need A Comparator To Compare The Numbers.

Here Is How We Can Implement It In Gate Level and Verilog Code.



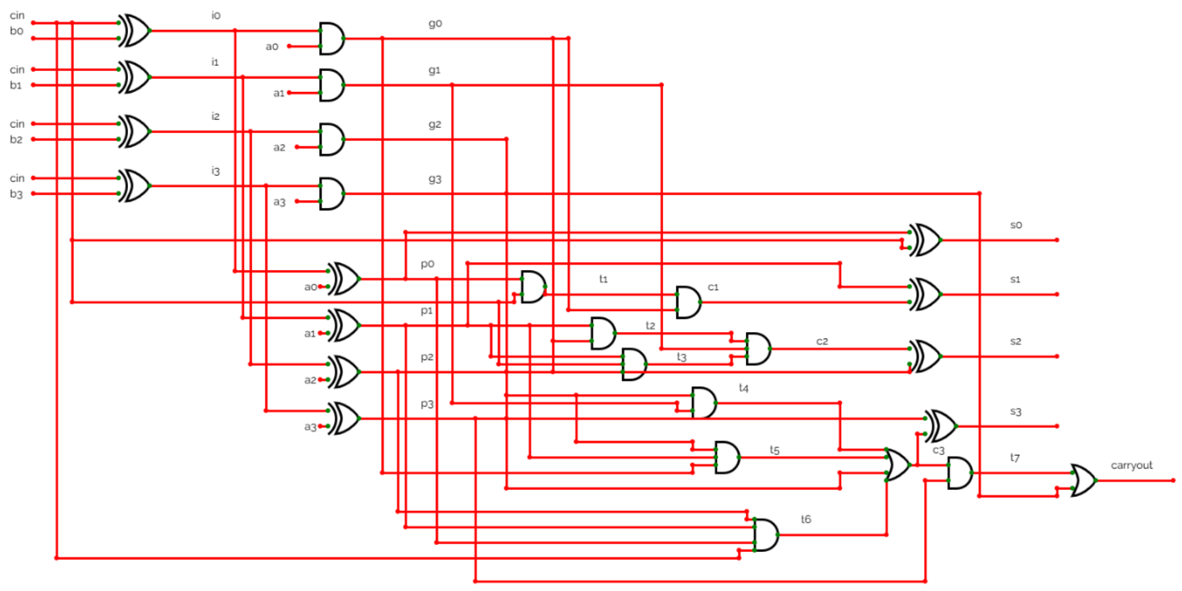
Here Is Our Comperator In Gate Level
And Below We Can See How We Implemented It In System Verilog



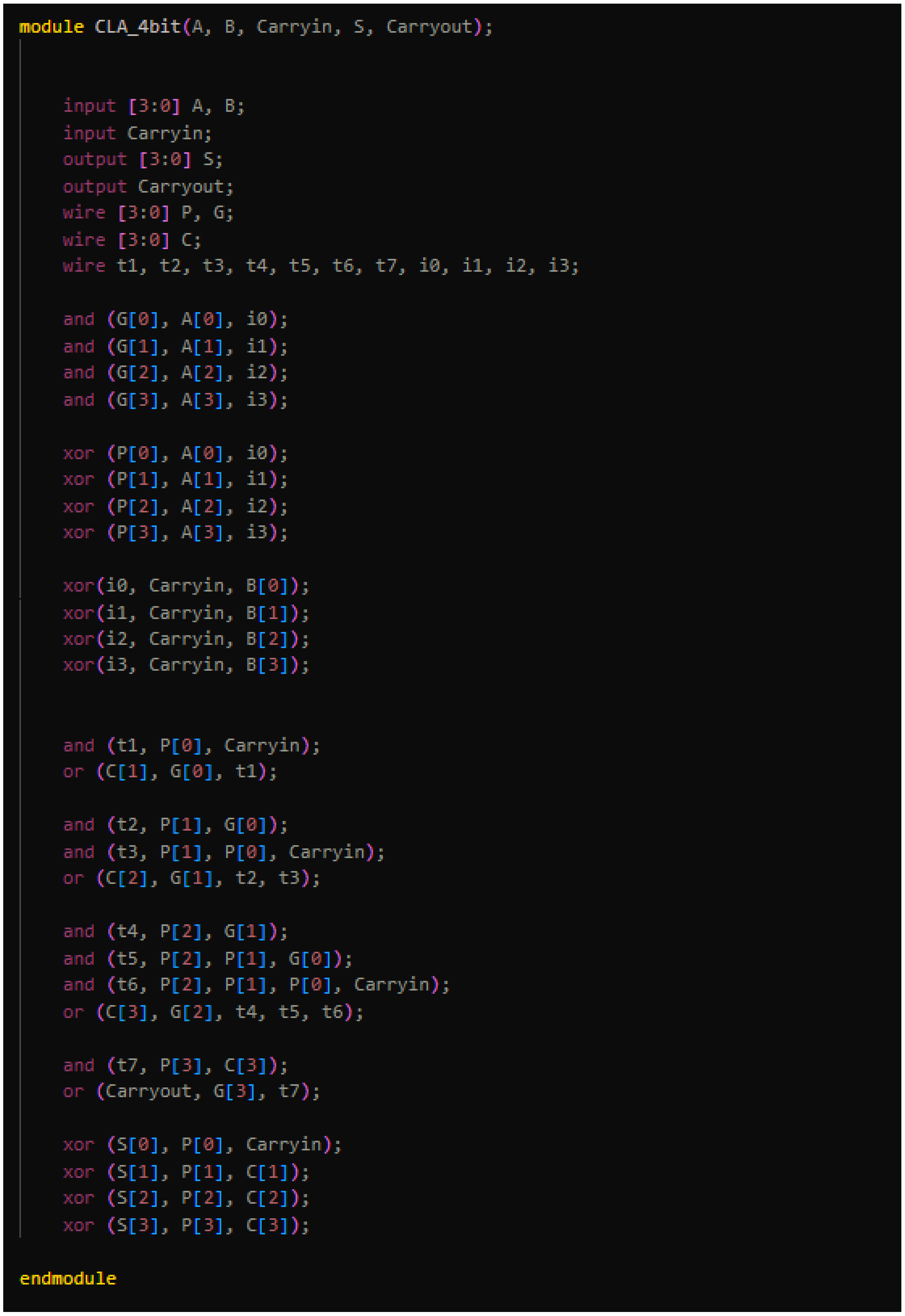
```
module Comperator(input [3:0] a, b, output E, AgB, BgA);
    wire i1, i2, i3, i4, i5, i6, i7, i8;
    wire NotB0, NotB1, NotB2, NotB3;

    xnor XNOR1(i1, a[0], b[0]);
    xnor XNOR2(i2, a[1], b[1]);
    xnor XNOR3(i3, a[2], b[2]);
    xnor XNOR4(i4, a[3], b[3]);

    not B3(NotB3, b[3]);
    not B2(NotB2, b[2]);
    not B1(NotB1, b[1]);
    not B0(NotB0, b[0]);

    and a1(i5, NotB3, a[3]);
    and a2(i6, NotB2, a[2], i3);
    and a3(i7, NotB1, i3, i2, a[1]);
    and a4(i8, NotB0, i3, i2, i1, a[0]);

    and Equal(E, i4, i3, i2, i1);
    or AgreaterThanB(AgB, i5, i6, i7, i8);

    nor (BgA, E, AgB);
endmodule
```

We Just Used The Comperator Logic And Implemented It Using Structural Method!

Then We Should Move To Our Next Module Which is Our Carry Look Ahead Adder Than Can Both Sub And Add Two Numbers Below You Can See The Gate Level And the Verilog Code Of it!



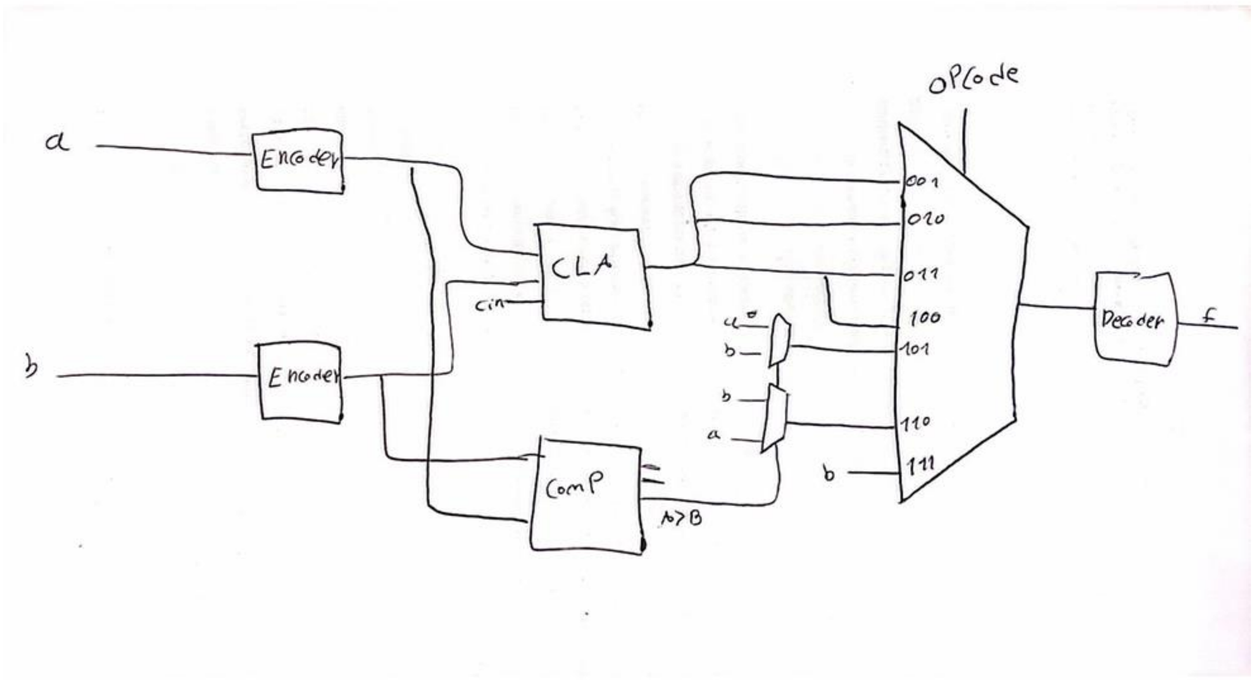
Here is The Gate Level Implementation Of CLA
And Below Is The Verilog Code Of It



```verilog
module CLA_4bit(A, B, Carryin, S, Carryout);



    input [3:0] A, B;
    input Carryin;
    output [3:0] S;
    output Carryout;
    wire [3:0] P, G;
    wire [3:0] C;
    wire t1, t2, t3, t4, t5, t6, t7, i0, i1, i2, i3;

    and (G[0], A[0], i0);
    and (G[1], A[1], i1);
    and (G[2], A[2], i2);
    and (G[3], A[3], i3);

    xor (P[0], A[0], i0);
    xor (P[1], A[1], i1);
    xor (P[2], A[2], i2);
    xor (P[3], A[3], i3);

    xor(i0, Carryin, B[0]);
    xor(i1, Carryin, B[1]);
    xor(i2, Carryin, B[2]);
    xor(i3, Carryin, B[3]);


    and (t1, P[0], Carryin);
    or (C[1], G[0], t1);

    and (t2, P[1], G[0]);
    and (t3, P[1], P[0], Carryin);
    or (C[2], G[1], t2, t3);

    and (t4, P[2], G[1]);
    and (t5, P[2], P[1], G[0]);
    and (t6, P[2], P[1], P[0], Carryin);
    or (C[3], G[2], t4, t5, t6);

    and (t7, P[3], C[3]);
    or (Carryout, G[3], t7);

    xor (S[0], P[0], Carryin);
    xor (S[1], P[1], C[1]);
    xor (S[2], P[2], C[2]);
    xor (S[3], P[3], C[3]);

endmodule
```

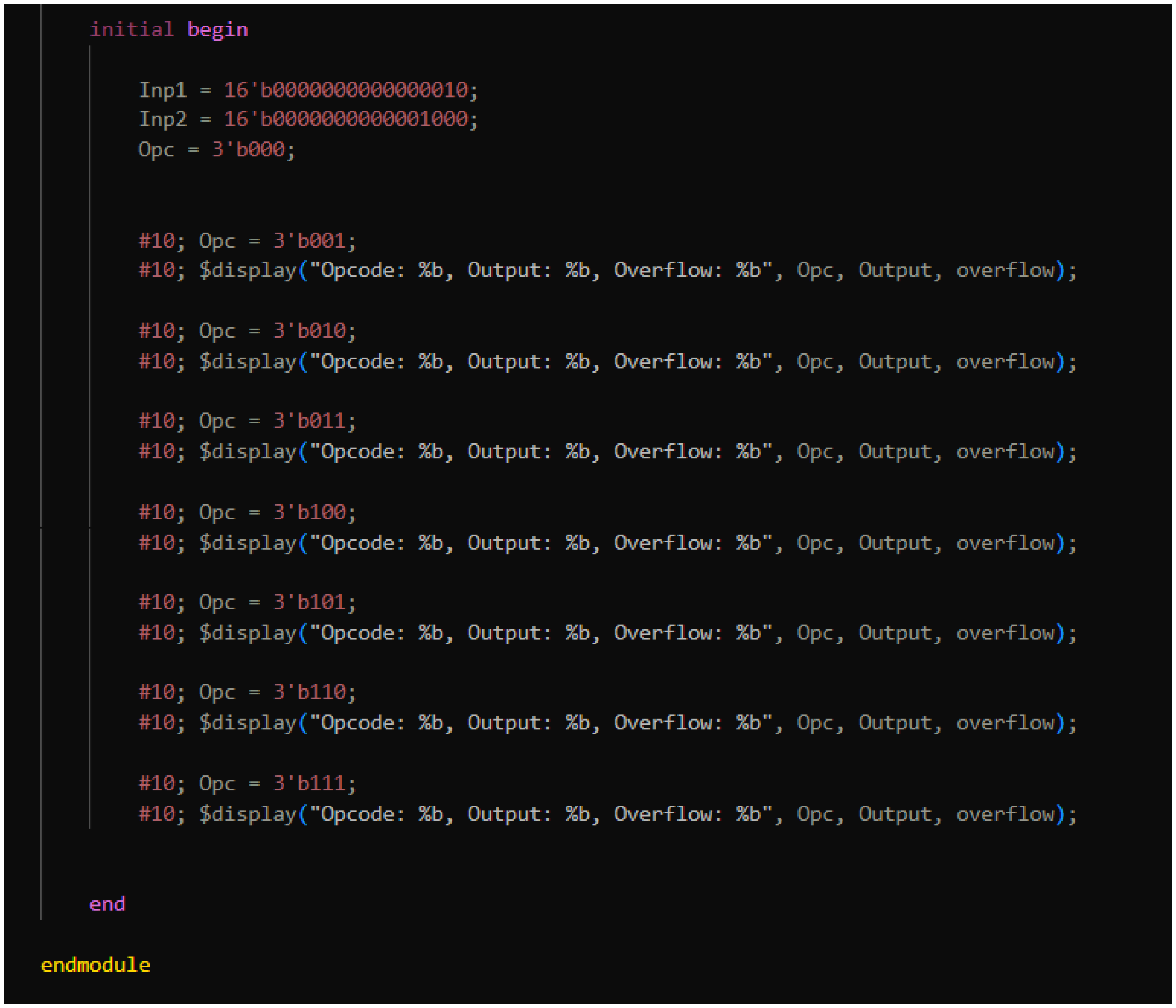
Next We Have To Implement Our ALU Which Kinda Represents A Multiplexer

For Every Opcode we Have To Decide So it It Showing Us The Multiplexer And The Picture Below Shows How Are Module IS Working



After All of These Now We Need To Check That If Our Code And Our Logic Is Correct Or Not So We Write A Test Bench For Our Top Module

Here Is My Test Best Bench That I Used



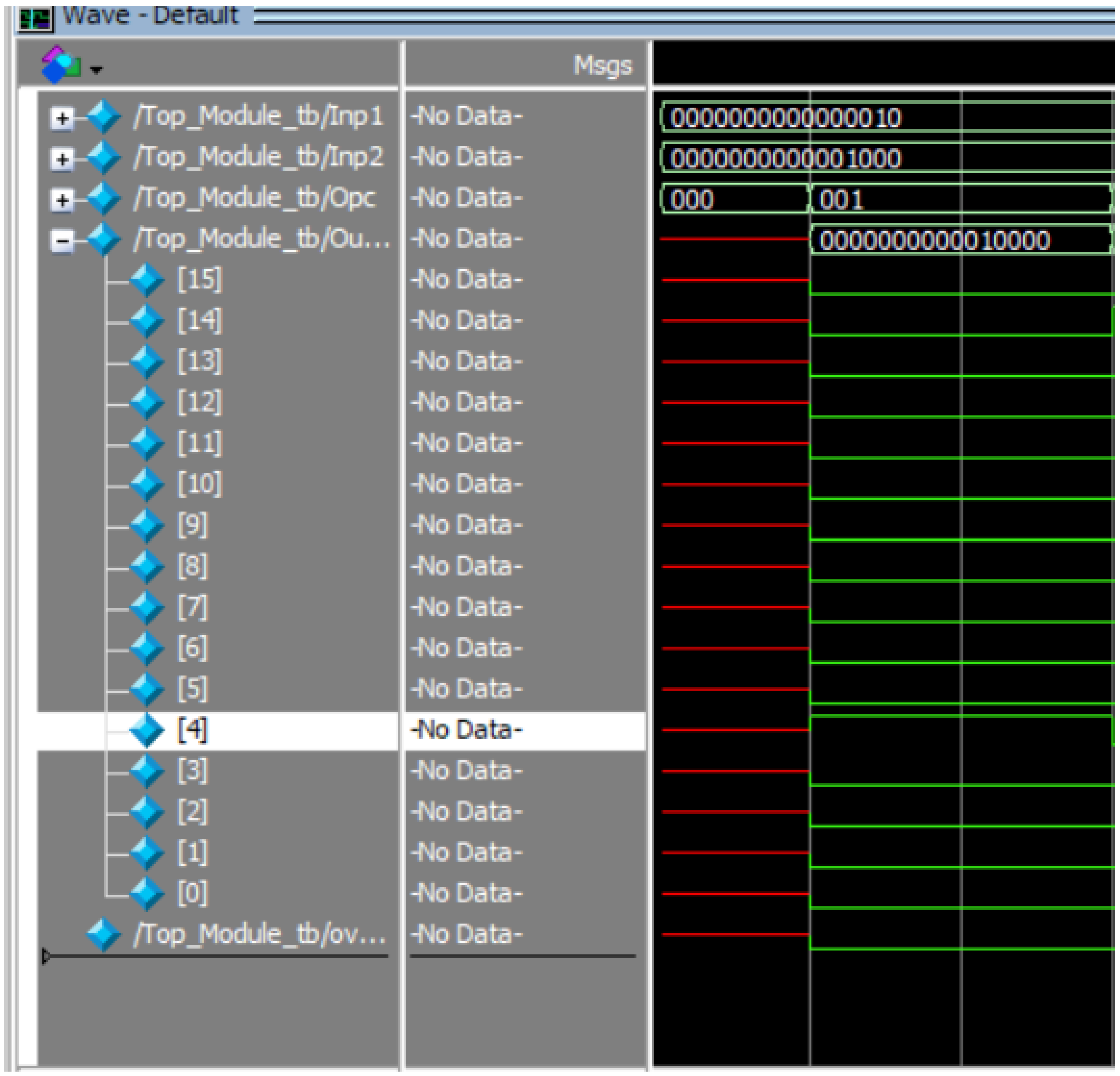
```
    initial begin

        Inp1 = 16'b0000000000000010;
        Inp2 = 16'b0000000000001000;
        Opc = 3'b000;


        #10; Opc = 3'b001;
        #10; $display("Opcode: %b, Output: %b, Overflow: %b", Opc, Output, overflow);

        #10; Opc = 3'b010;
        #10; $display("Opcode: %b, Output: %b, Overflow: %b", Opc, Output, overflow);

        #10; Opc = 3'b011;
        #10; $display("Opcode: %b, Output: %b, Overflow: %b", Opc, Output, overflow);

        #10; Opc = 3'b100;
        #10; $display("Opcode: %b, Output: %b, Overflow: %b", Opc, Output, overflow);

        #10; Opc = 3'b101;
        #10; $display("Opcode: %b, Output: %b, Overflow: %b", Opc, Output, overflow);

        #10; Opc = 3'b110;
        #10; $display("Opcode: %b, Output: %b, Overflow: %b", Opc, Output, overflow);

        #10; Opc = 3'b111;
        #10; $display("Opcode: %b, Output: %b, Overflow: %b", Opc, Output, overflow);


    end

endmodule
```

I Used Inp1 = 1 And Inp2 = 3

Our Next Step Is To Look At The Wave Form Of Them And Check

At First Our Opcode is 0 That Is Default For It

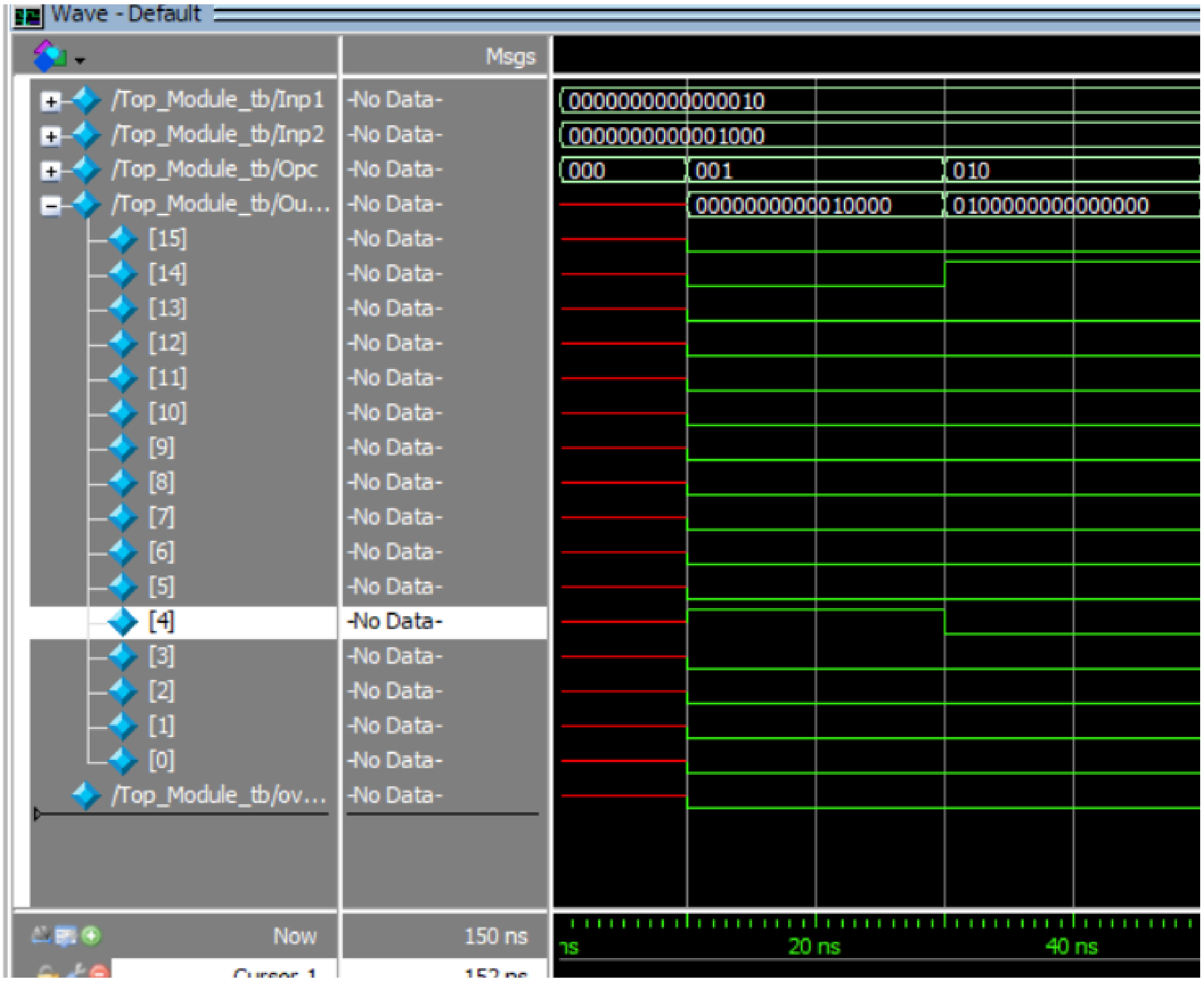
After It We Move To 001 Which Is Addition That If We Add 1 + 3 = 4 And its 0000000000001000 that means 4 and its correct



Wave - Default

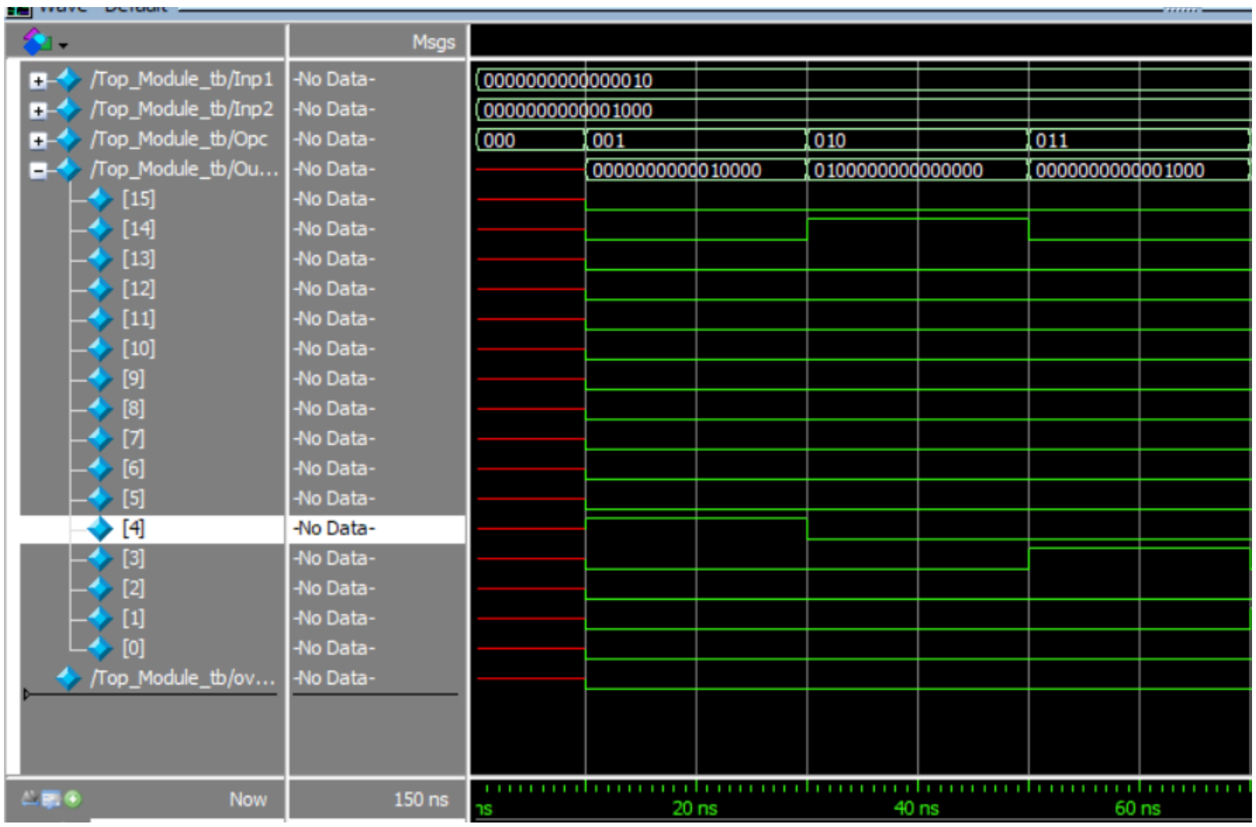
	Msgs	
/Top_Module_tb/Inp1	-No Data-	0000000000000010
/Top_Module_tb/Inp2	-No Data-	0000000000001000
/Top_Module_tb/Opc	-No Data-	000 001
/Top_Module_tb/Ou...	-No Data-	0000000000010000
[15]	-No Data-	
[14]	-No Data-	
[13]	-No Data-	
[12]	-No Data-	
[11]	-No Data-	
[10]	-No Data-	
[9]	-No Data-	
[8]	-No Data-	
[7]	-No Data-	
[6]	-No Data-	
[5]	-No Data-	
[4]	-No Data-	
[3]	-No Data-	
[2]	-No Data-	
[1]	-No Data-	
[0]	-No Data-	
/Top_Module_tb/ov...	-No Data-	

After That We Move To Sub



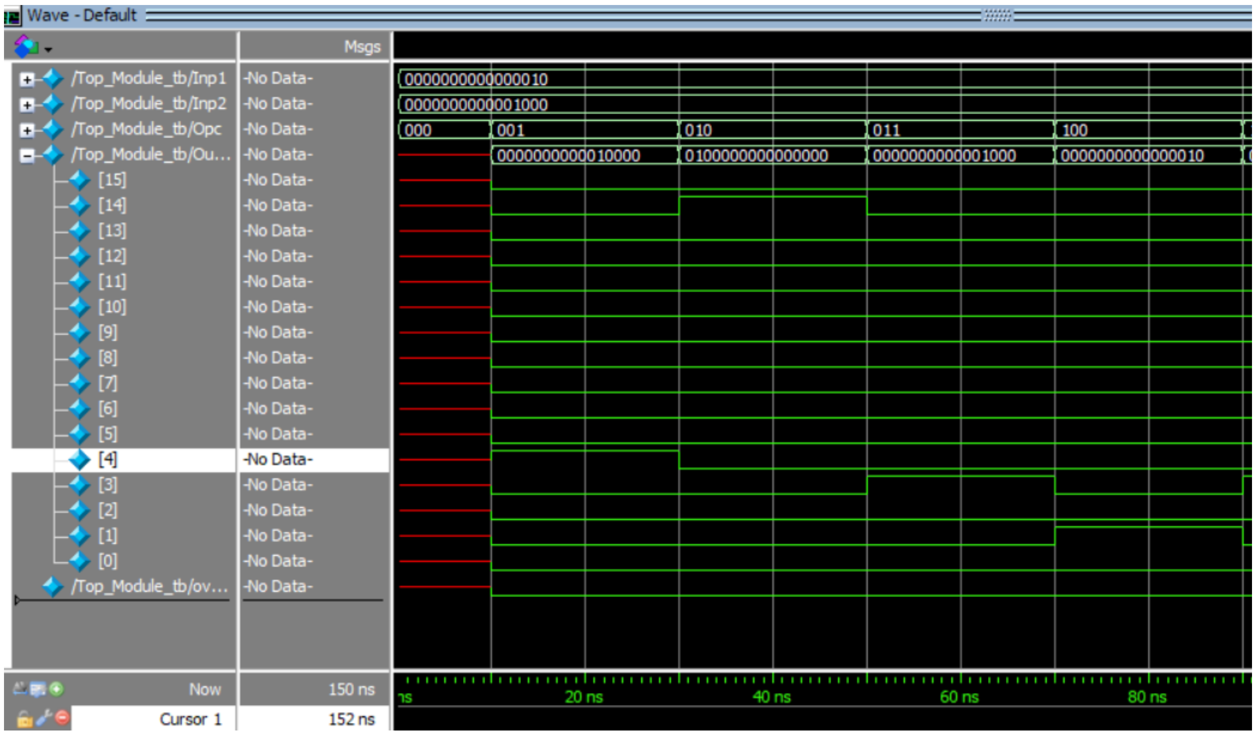
In this case Our first input is 1 and second is 3 so the Result should be -2 wich is showing right and 010000000000000000 is showing -2

Next We Want to Compare these two numbers and find the maximum of it



011 showing the code and we want to find the Max(1,3) which is 3 and showing correct in the wave form

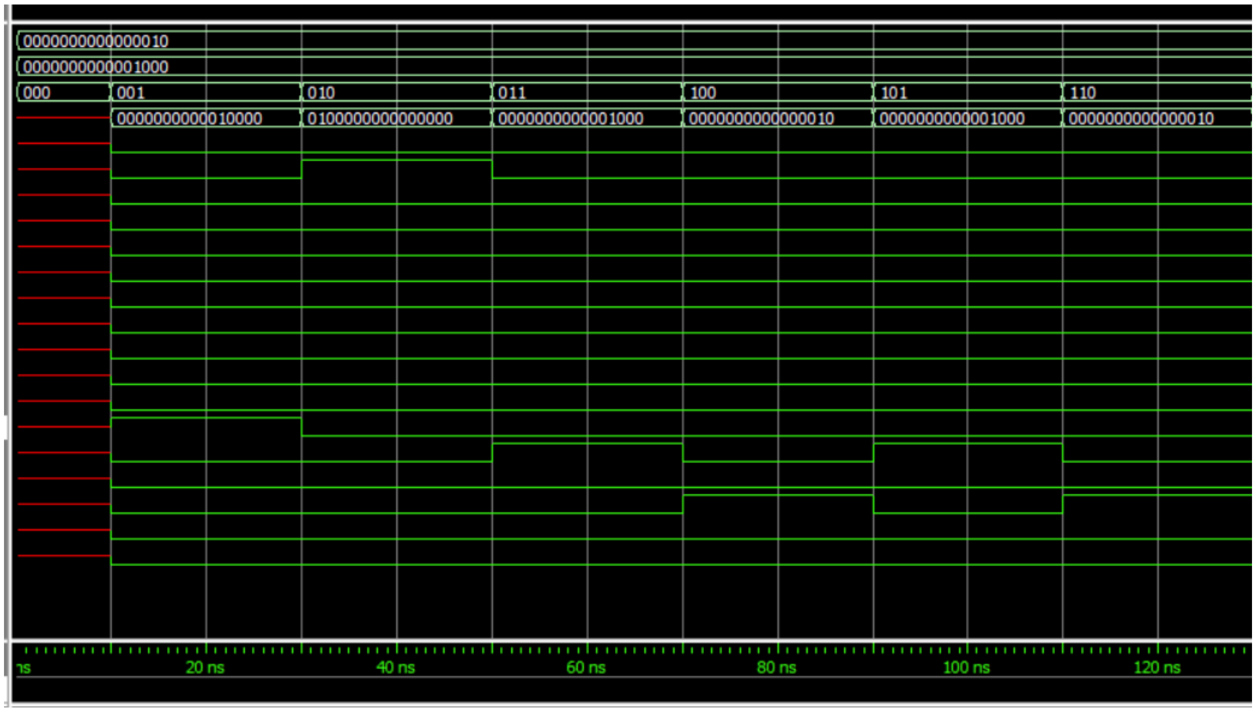
Then We Want to Find The minimum of it that our code is 100



and the Min(1,3) = 1 Which is correct in the wave form

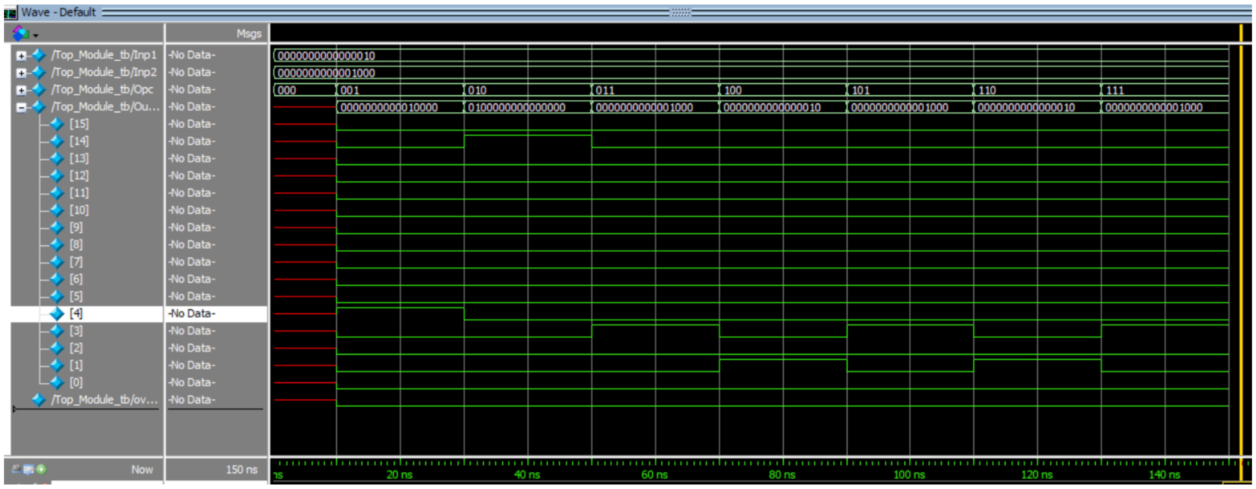
Next Step We Want to Compare These Numbers Now With Comparator

That We Can See Its Correct



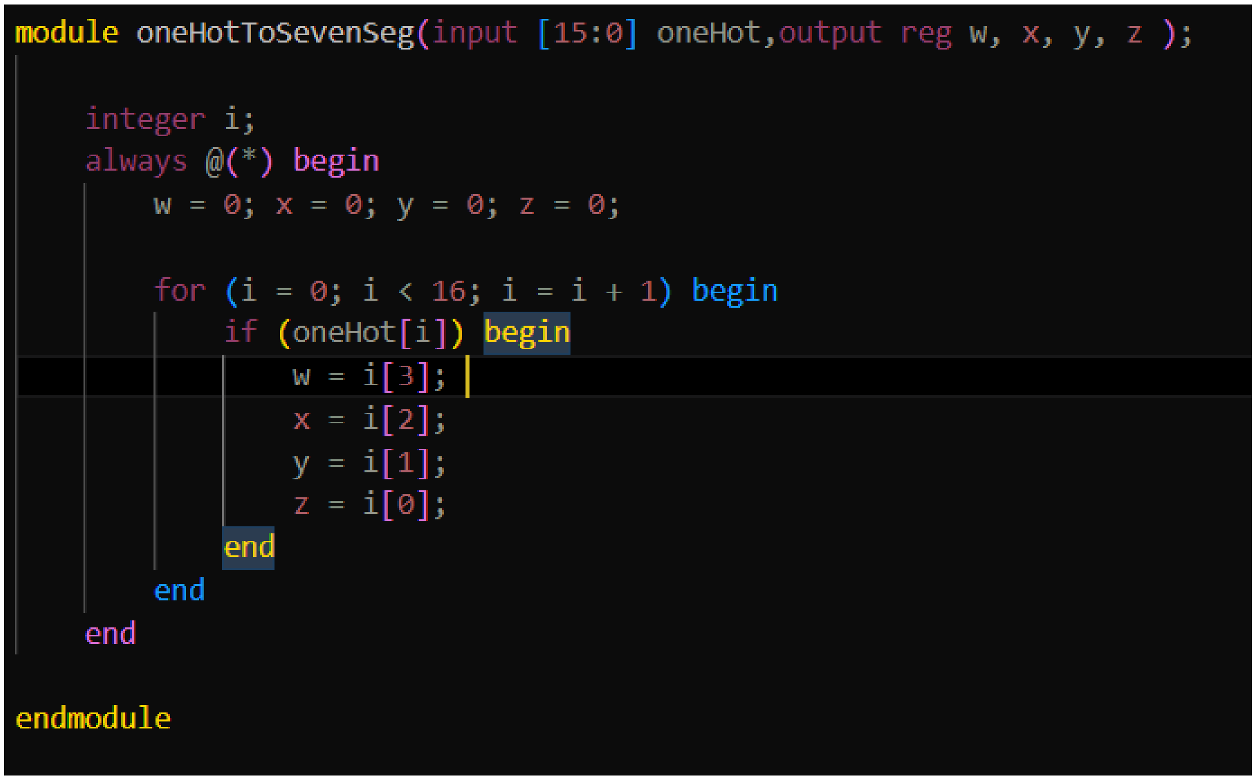
----

and for the last step we need to return the second input which represent the code 0f 111



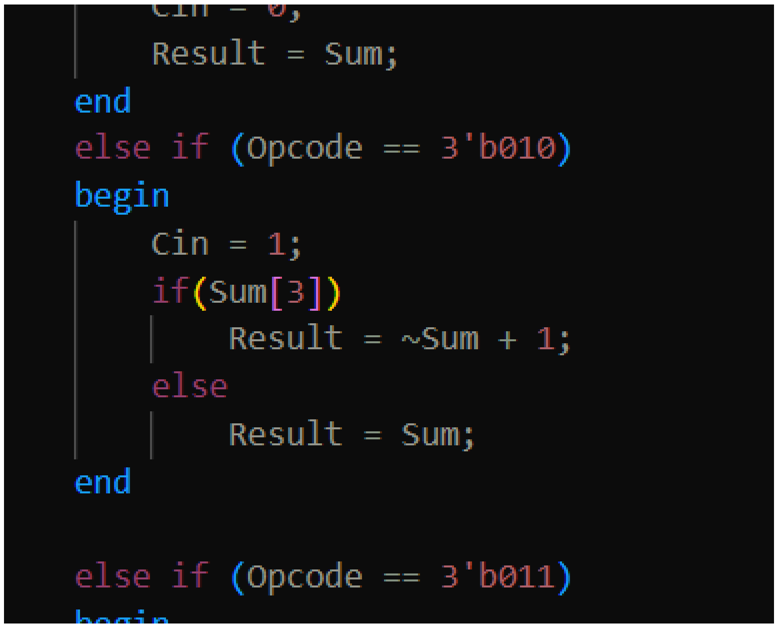
and here is the Overall View For The Code That We Can See All The Modules Are Working Correct!

For Showing Seven Segment On Our Output We Have To Make A Module That Converts A OneHot To Seven Segment Number Which Can Implement Like This



```
module oneHotToSevenSeg(input [15:0] oneHot,output reg w, x, y, z );

    integer i;
    always @(*) begin
        w = 0; x = 0; y = 0; z = 0;

        for (i = 0; i < 16; i = i + 1) begin
            if (oneHot[i]) begin
                w = i[3];
                x = i[2];
                y = i[1];
                z = i[0];
            end
        end
    end

endmodule
```

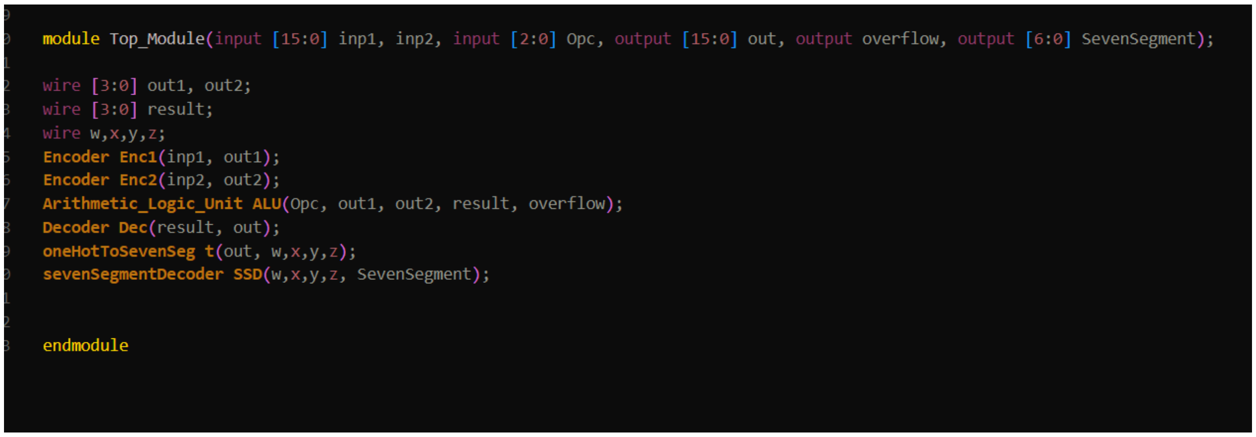
After That We Need To Handle That IF Our Sum Weren`t Positive We Convert Our Number Into Twos` Complement



```
        Cin = 0;
        Result = Sum;
    end
    else if (Opcode == 3'b010)
    begin
        Cin = 1;
        if(Sum[3])
            Result = ~Sum + 1;
        else
            Result = Sum;
    end

    else if (Opcode == 3'b011)
    begin
```

And For Last Thing We Need To Make Some Changes To Our Top Module

`

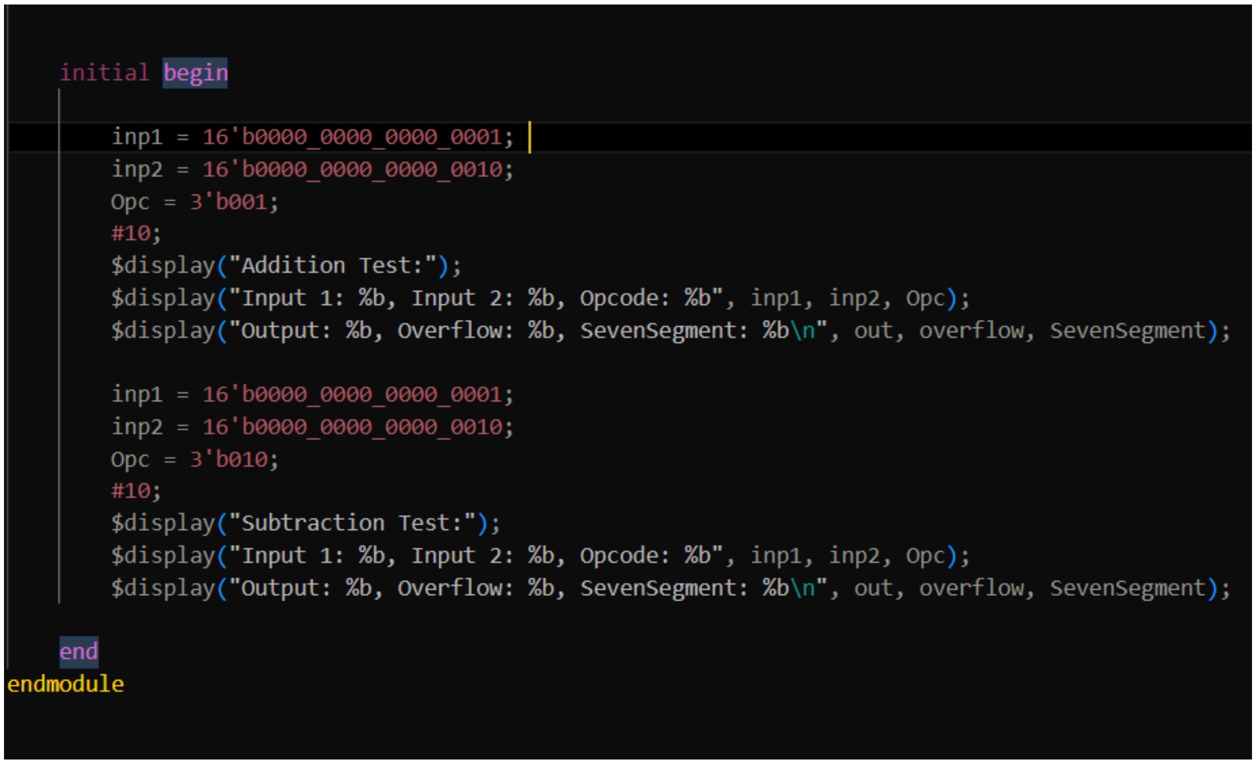


```
module Top_Module(input [15:0] inp1, inp2, input [2:0] Opc, output [15:0] out, output overflow, output [6:0] SevenSegment);

wire [3:0] out1, out2;
wire [3:0] result;
wire w,x,y,z;
Encoder Enc1(inp1, out1);
Encoder Enc2(inp2, out2);
Arithmetic_Logic_Unit ALU(Opc, out1, out2, result, overflow);
Decoder Dec(result, out);
oneHotToSevenSeg t(out, w,x,y,z);
sevenSegmentDecoder SSD(w,x,y,z, SevenSegment);


endmodule
```

Changed the output to 7 bit and added some wires for w,x,y,z and a 7 segment output for showing results

Now lets test it

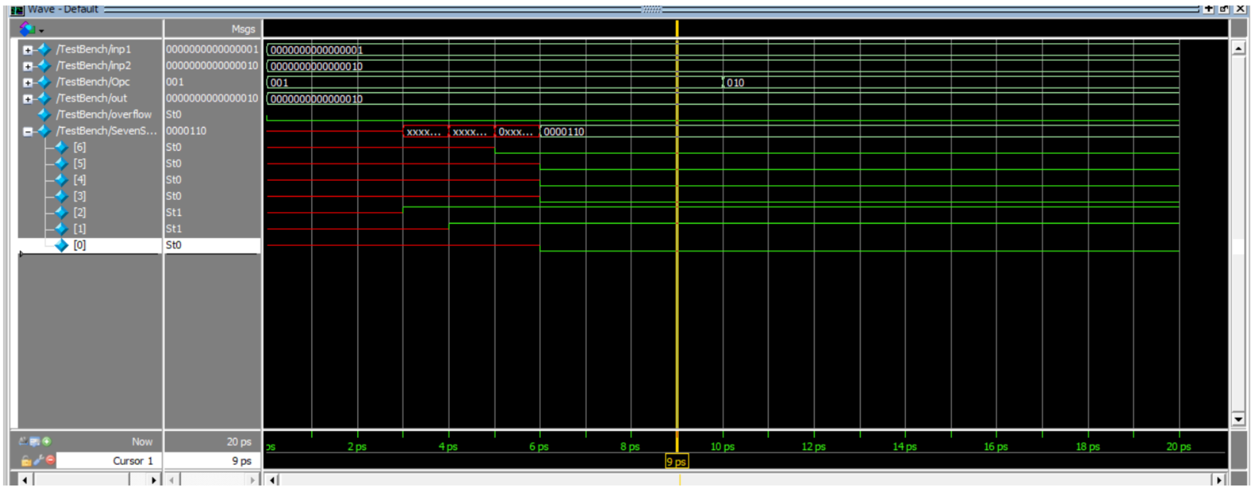
Here is My test Bench for Seven Segment



```
    initial begin

        inp1 = 16'b0000_0000_0000_0001;
        inp2 = 16'b0000_0000_0000_0010;
        Opc = 3'b001;
        #10;
        $display("Addition Test:");
        $display("Input 1: %b, Input 2: %b, Opcode: %b", inp1, inp2, Opc);
        $display("Output: %b, Overflow: %b, SevenSegment: %b\n", out, overflow, SevenSegment);

        inp1 = 16'b0000_0000_0000_0001;
        inp2 = 16'b0000_0000_0000_0010;
        Opc = 3'b010;
        #10;
        $display("Subtraction Test:");
        $display("Input 1: %b, Input 2: %b, Opcode: %b", inp1, inp2, Opc);
        $display("Output: %b, Overflow: %b, SevenSegment: %b\n", out, overflow, SevenSegment);

    end
endmodule
```

For this test first I used 0 and 1 for addition

And for second test I used the same numbers but this time for subtraction
as you can see the result at first test is 1 and second is -1 but we chaged the -1 to absolute to show it on seven segment which you can see both are correct



	Msgs
/TestBench/inp1	0000000000000001
/TestBench/inp2	0000000000000010
/TestBench/Opc	001
/TestBench/out	0000000000000010
/TestBench/overflow	St0
/TestBench/SevenS...	0000110
[6]	St0
[5]	St0
[4]	St0
[3]	St0
[2]	St1
[1]	St1
[0]	St0

0000110 is representing 1 in seven segment