Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Engineering
Department of Electronic and Electrical Engineering

3C7 – Project 1

Mini-Arithmetic Logic Unit

3C7 – Digital Systems Design

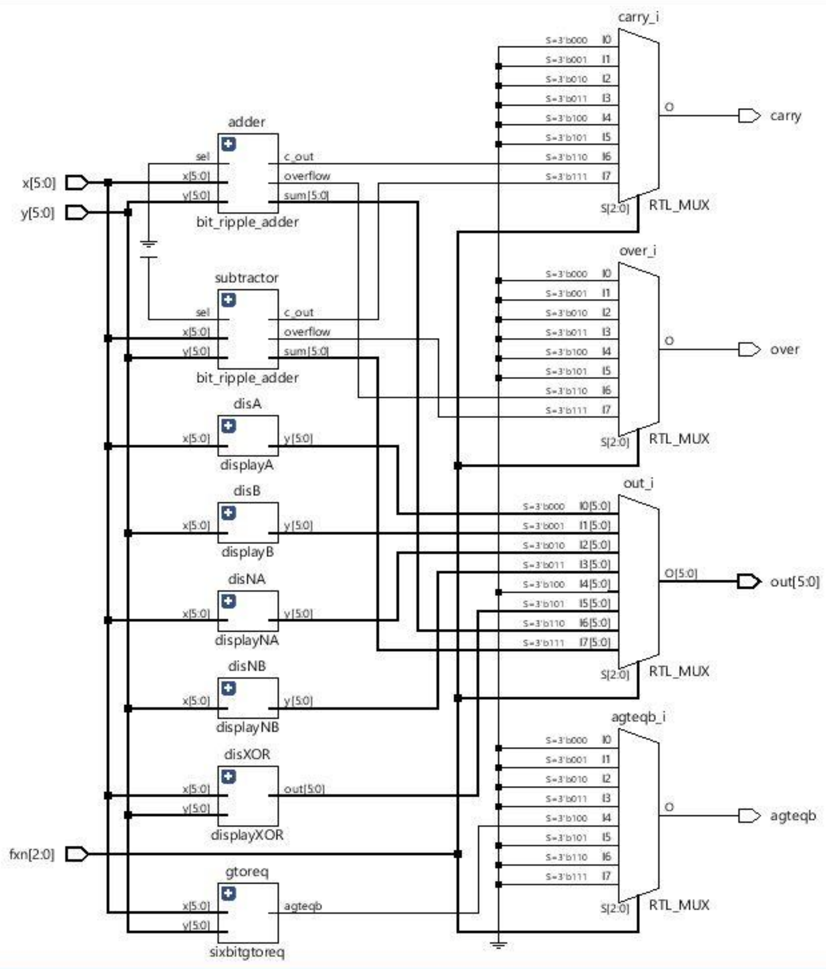
Name: Videept Kohli
Student Number:18335157
Date of Submission: 14/03/19
Lecturer: Naomi Harte

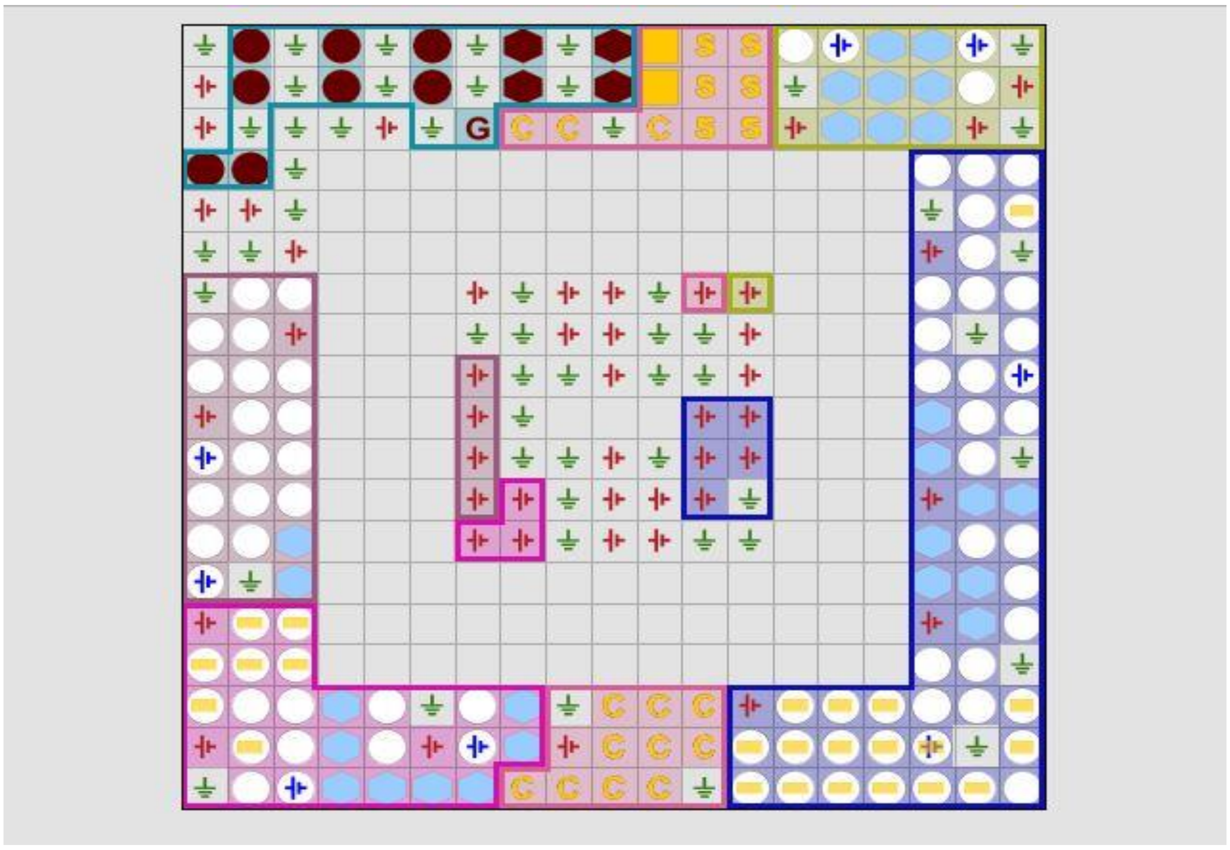
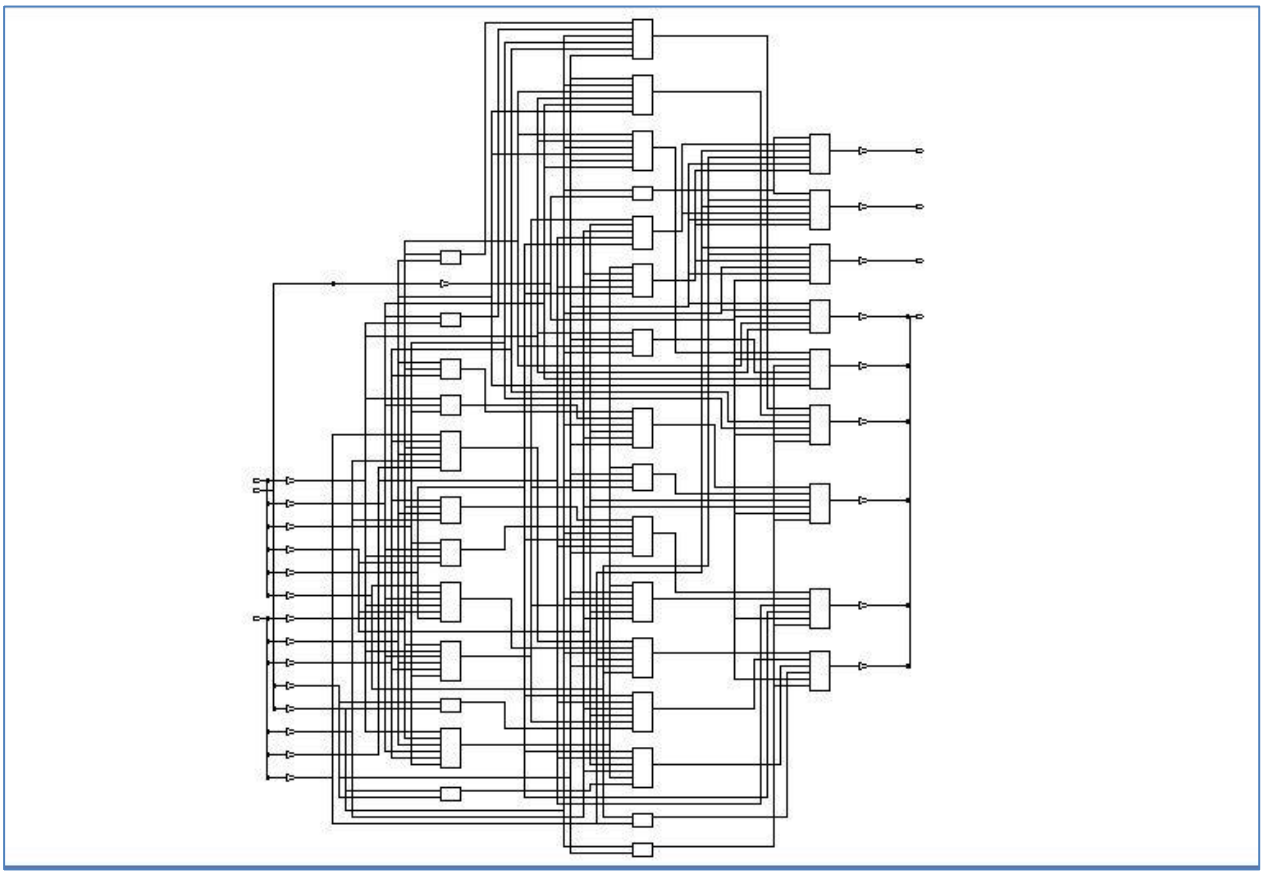
Aim:

You need to design, write/modify the Verilog modules for, and test a “mini” arithmetic logic unit. An arithmetic logic unit (ALU) performs combinatorial logic, implementing arithmetic functions. A typical ALU has a wide range of functionality from addition to bit shifting. Your ALU will provide a narrow range of functions performed on two 6-bit inputs A and B. A and B are in 2’s complement format. The output of the ALU is a 6-bit number X, also in 2’s complement form as appropriate, and the input fxn controls the output as follows:

fxn	X[5:0]
000	**A**
001	**B**
010	**-A**
011	**-B**
100	**A>=B (is A greater than or equal to B)**
101	**A^B (Bitwise exclusive OR)**
110	**A+B**
111	**A-B**

Functional Block Diagrams:



SECTION 1: TESTBENCH WAVEFORMS

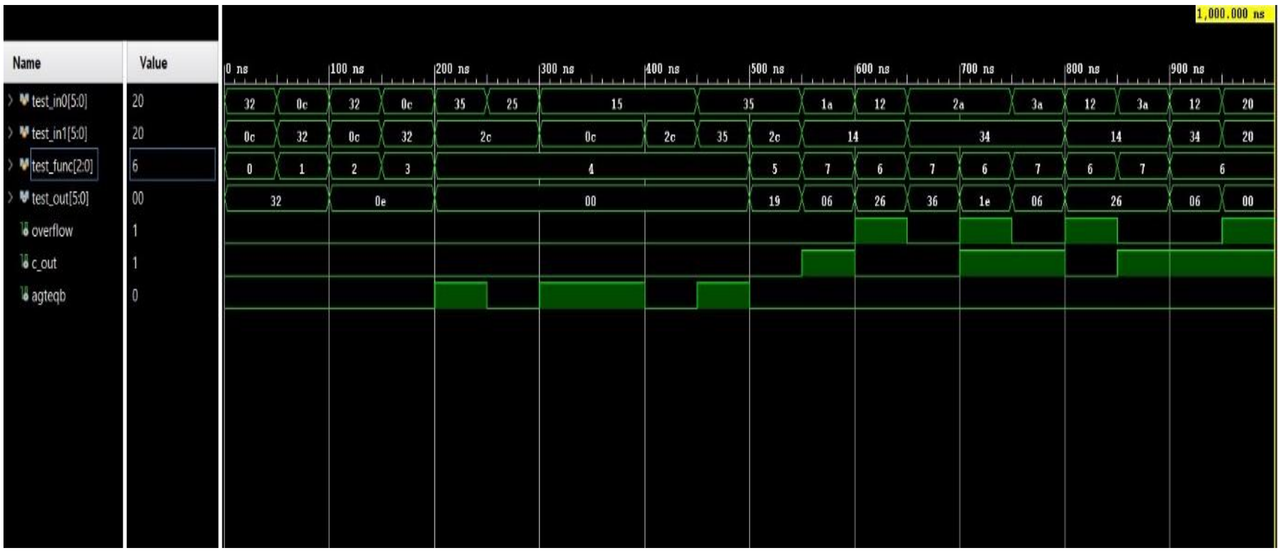


Fig 1.1: Compressed Form of Testbench

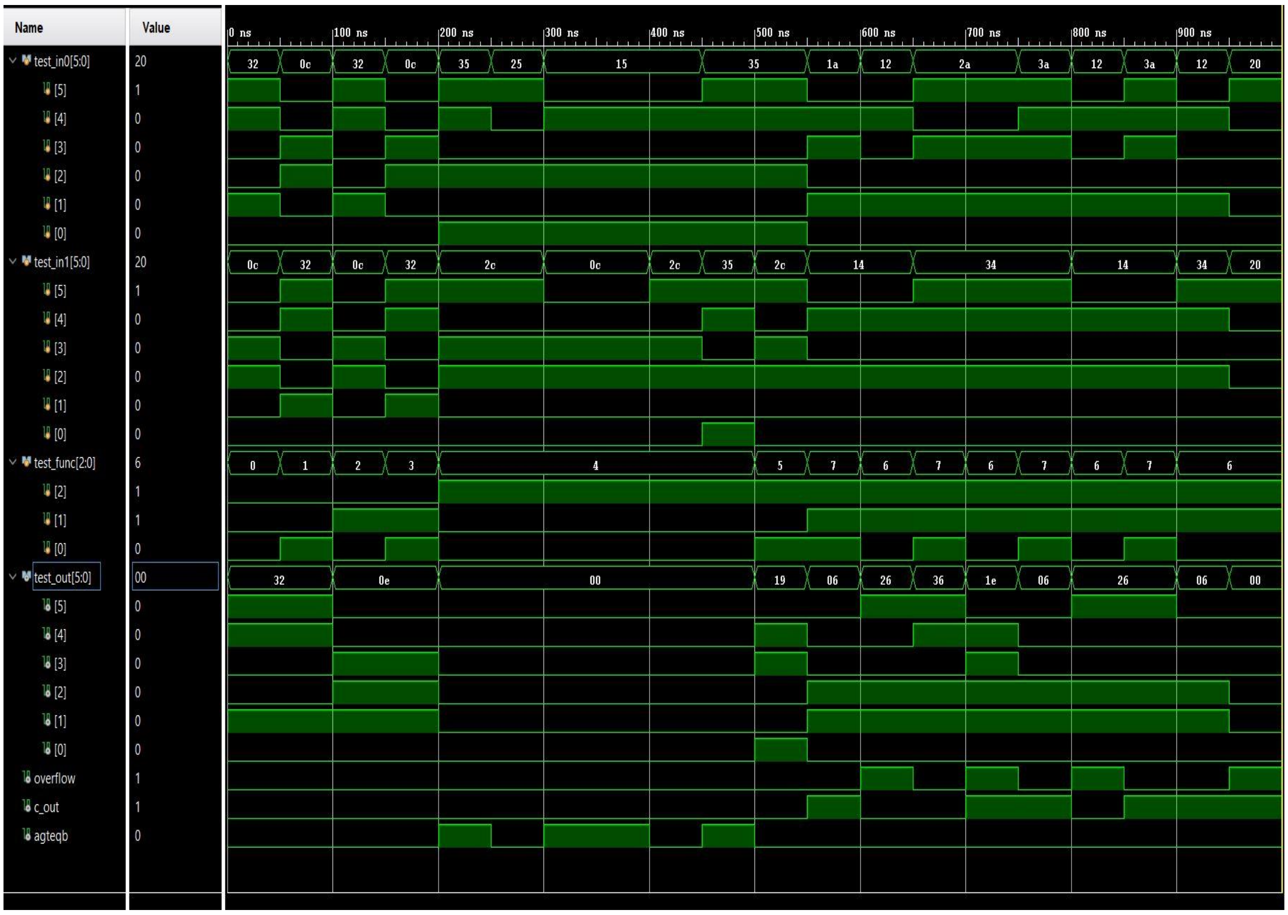
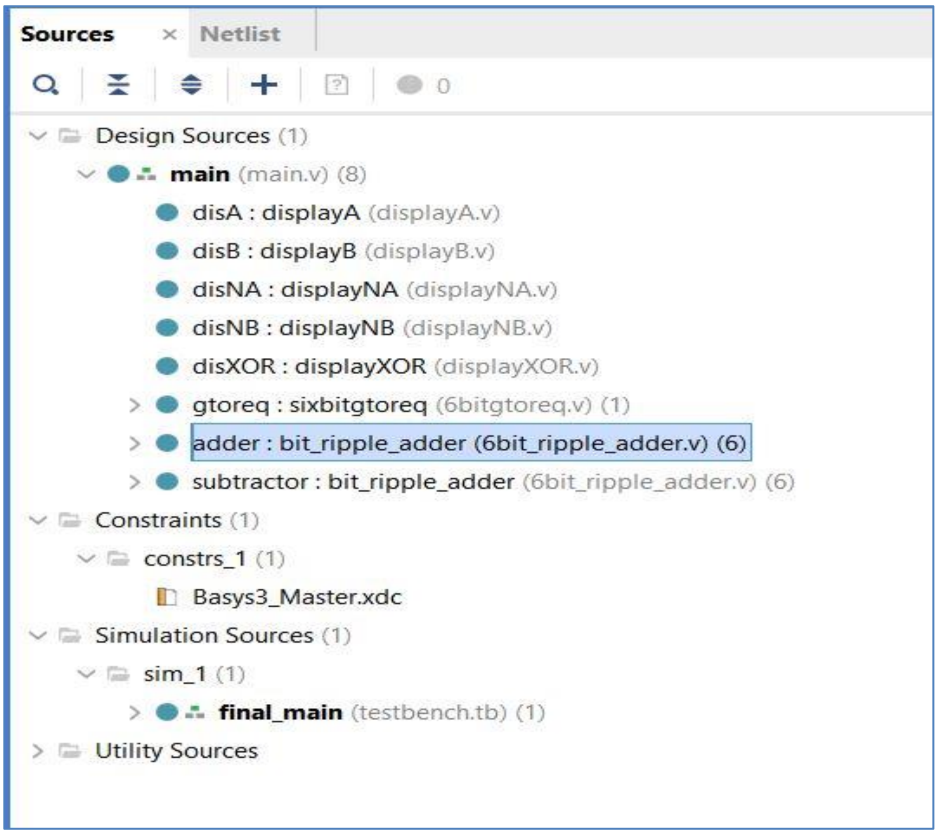


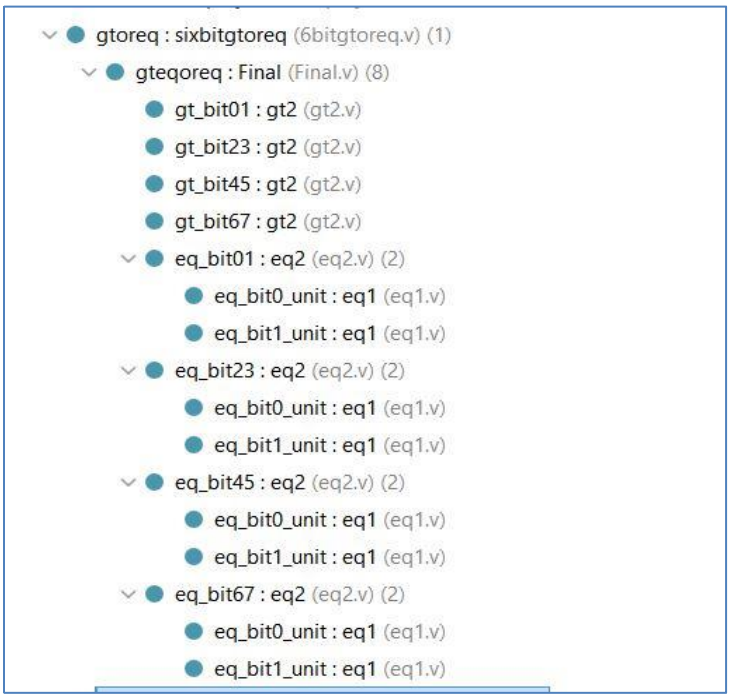
Fig 1.2: Expanded Form of Testbench

SECTION 2: HIERARCHY OF VIVADO DESIGN SOURCES



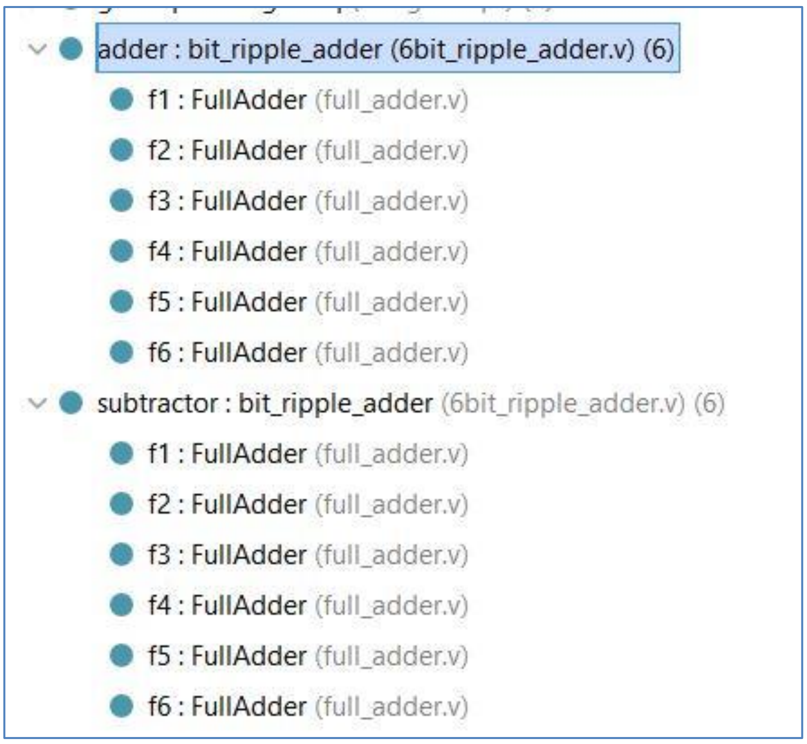
```
Sources  ×  Netlist
0
Design Sources (1)
  main (main.v) (8)
    disA : displayA (displayA.v)
    disB : displayB (displayB.v)
    disNA : displayNA (displayNA.v)
    disNB : displayNB (displayNB.v)
    disXOR : displayXOR (displayXOR.v)
    gtoreq : sixbitgtoreq (6bitgtoreq.v) (1)
    adder : bit_ripple_adder (6bit_ripple_adder.v) (6)
    subtractor : bit_ripple_adder (6bit_ripple_adder.v) (6)
Constraints (1)
  constrs_1 (1)
    Basys3_Master.xdc
Simulation Sources (1)
  sim_1 (1)
    final_main (testbench.tb) (1)
Utility Sources
```

FIGURE 2.1: Compressed form of Sources



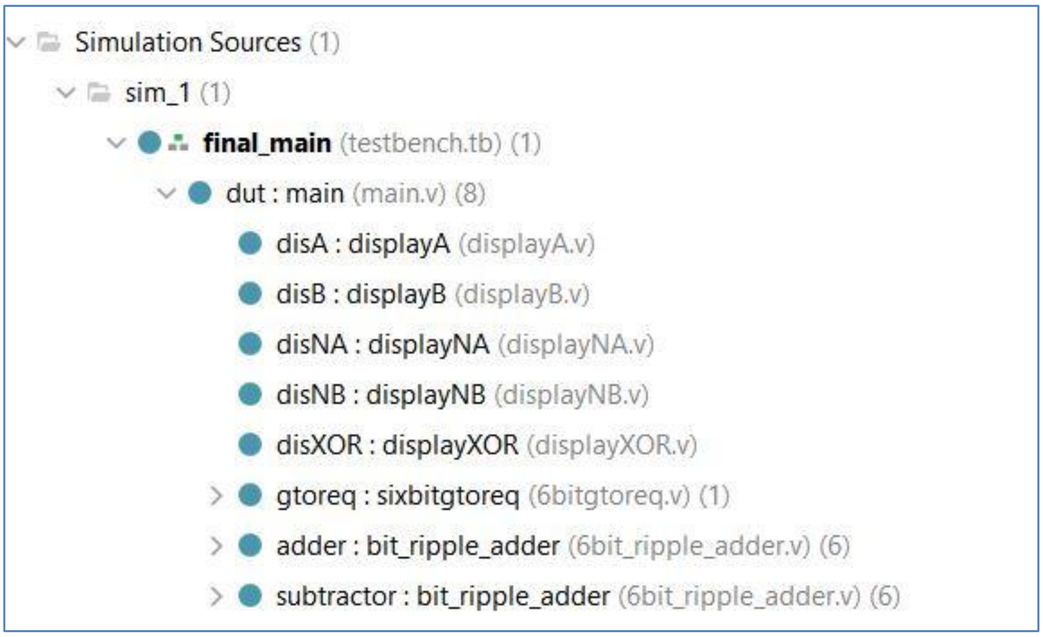
```
gtoreq : sixbitgtoreq (6bitgtoreq.v) (1)
  gteqoreq : Final (Final.v) (8)
    gt_bit01 : gt2 (gt2.v)
    gt_bit23 : gt2 (gt2.v)
    gt_bit45 : gt2 (gt2.v)
    gt_bit67 : gt2 (gt2.v)
    eq_bit01 : eq2 (eq2.v) (2)
      eq_bit0_unit : eq1 (eq1.v)
      eq_bit1_unit : eq1 (eq1.v)
    eq_bit23 : eq2 (eq2.v) (2)
      eq_bit0_unit : eq1 (eq1.v)
      eq_bit1_unit : eq1 (eq1.v)
    eq_bit45 : eq2 (eq2.v) (2)
      eq_bit0_unit : eq1 (eq1.v)
      eq_bit1_unit : eq1 (eq1.v)
    eq_bit67 : eq2 (eq2.v) (2)
      eq_bit0_unit : eq1 (eq1.v)
      eq_bit1_unit : eq1 (eq1.v)
```

FIGURE 2.2: Expanded Form of 6 Bit Greater Than or Equal To



```
⌄ ● adder : bit_ripple_adder (6bit_ripple_adder.v) (6)
      ● f1 : FullAdder (full_adder.v)
      ● f2 : FullAdder (full_adder.v)
      ● f3 : FullAdder (full_adder.v)
      ● f4 : FullAdder (full_adder.v)
      ● f5 : FullAdder (full_adder.v)
      ● f6 : FullAdder (full_adder.v)
⌄ ● subtractor : bit_ripple_adder (6bit_ripple_adder.v) (6)
      ● f1 : FullAdder (full_adder.v)
      ● f2 : FullAdder (full_adder.v)
      ● f3 : FullAdder (full_adder.v)
      ● f4 : FullAdder (full_adder.v)
      ● f5 : FullAdder (full_adder.v)
      ● f6 : FullAdder (full_adder.v)
```

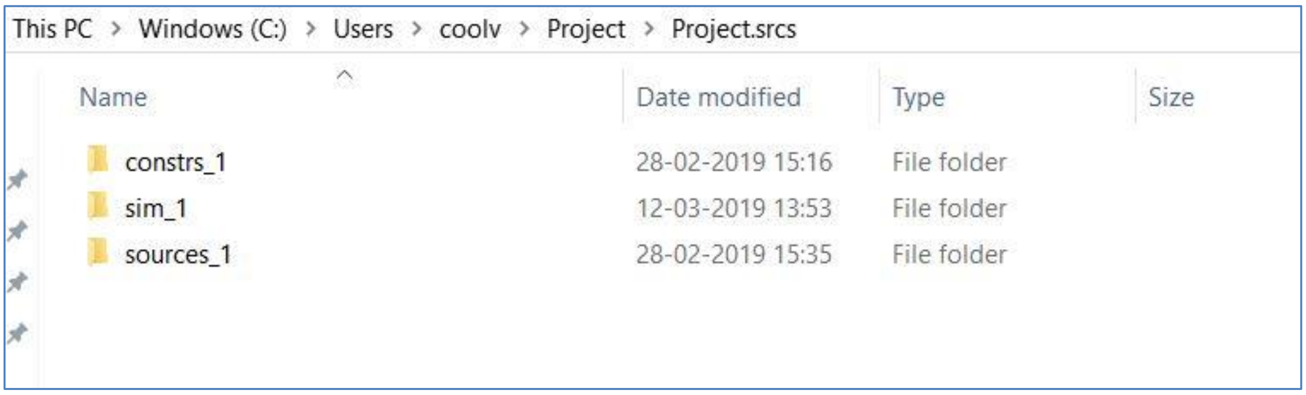
FIGURE 2.3: Expanded Form of Adder Subtractor



```
⌄ Simulation Sources (1)
   ⌄ sim_1 (1)
      ⌄ ● final_main (testbench.tb) (1)
         ⌄ ● dut : main (main.v) (8)
               ● disA : displayA (displayA.v)
               ● disB : displayB (displayB.v)
               ● disNA : displayNA (displayNA.v)
               ● disNB : displayNB (displayNB.v)
               ● disXOR : displayXOR (displayXOR.v)
             > ● gtoreq : sixbitgtoreq (6bitgtoreq.v) (1)
             > ● adder : bit_ripple_adder (6bit_ripple_adder.v) (6)
             > ● subtractor : bit_ripple_adder (6bit_ripple_adder.v) (6)
```

FIGURE 2.4: Expanded Form of Testbench Hierarchy

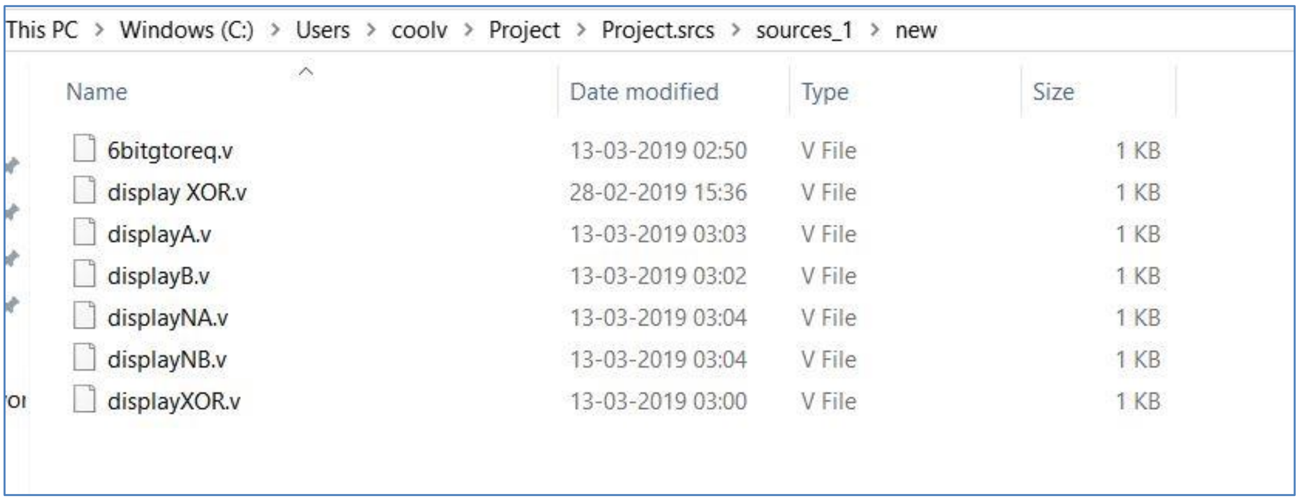
SECTION 3: FILE DIRECTORY STRUCTURE



This PC > Windows (C:) > Users > coolv > Project > Project.srcs

Name	Date modified	Type	Size
constrs_1	28-02-2019 15:16	File folder	
sim_1	12-03-2019 13:53	File folder	
sources_1	28-02-2019 15:35	File folder	

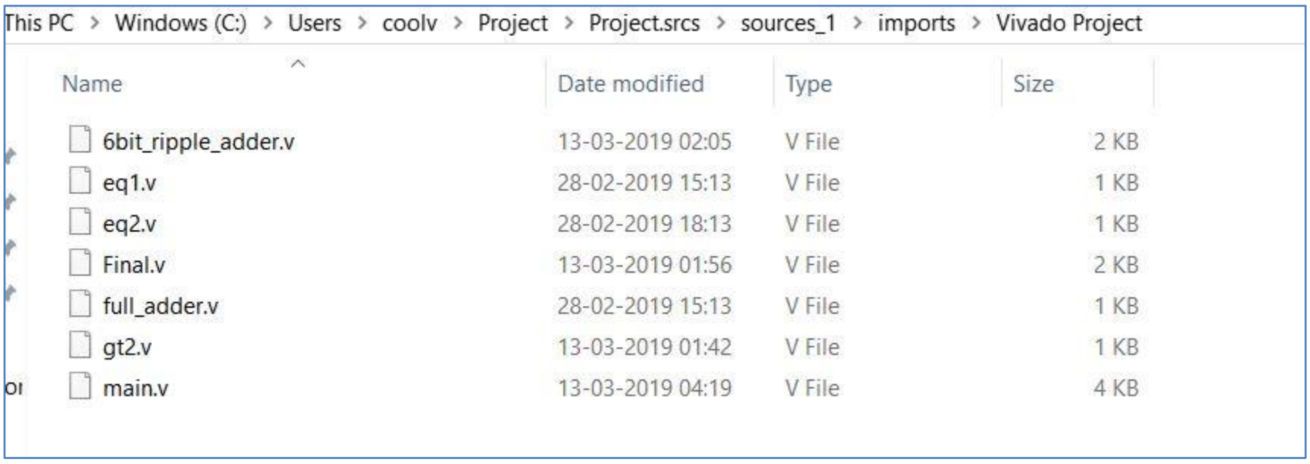
FIGURE 3.1: Compressed Form of File Directory Structure



This PC > Windows (C:) > Users > coolv > Project > Project.srcs > sources_1 > new

Name	Date modified	Type	Size
6bitgtoreq.v	13-03-2019 02:50	V File	1 KB
display XOR.v	28-02-2019 15:36	V File	1 KB
displayA.v	13-03-2019 03:03	V File	1 KB
displayB.v	13-03-2019 03:02	V File	1 KB
displayNA.v	13-03-2019 03:04	V File	1 KB
displayNB.v	13-03-2019 03:04	V File	1 KB
displayXOR.v	13-03-2019 03:00	V File	1 KB

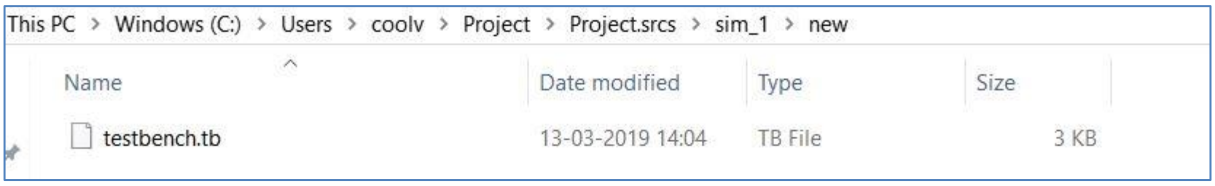
FIGURE 3.2: Expanded Form of New Sources



This PC > Windows (C:) > Users > coolv > Project > Project.srcs > sources_1 > imports > Vivado Project

Name	Date modified	Type	Size
6bit_ripple_adder.v	13-03-2019 02:05	V File	2 KB
eq1.v	28-02-2019 15:13	V File	1 KB
eq2.v	28-02-2019 18:13	V File	1 KB
Final.v	13-03-2019 01:56	V File	2 KB
full_adder.v	28-02-2019 15:13	V File	1 KB
gt2.v	13-03-2019 01:42	V File	1 KB
main.v	13-03-2019 04:19	V File	4 KB

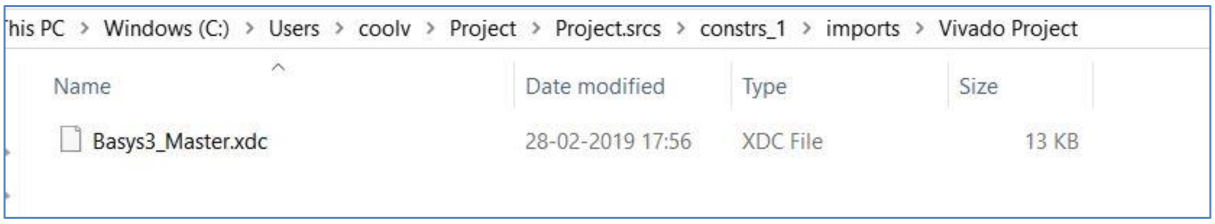
FIGURE 3.3: Expanded Form of Sources Imported



This PC > Windows (C:) > Users > coolv > Project > Project.srcs > sim_1 > new

Name	Date modified	Type	Size
testbench.tb	13-03-2019 14:04	TB File	3 KB

FIGURE 3.4: Expanded Form of Simulation Sources



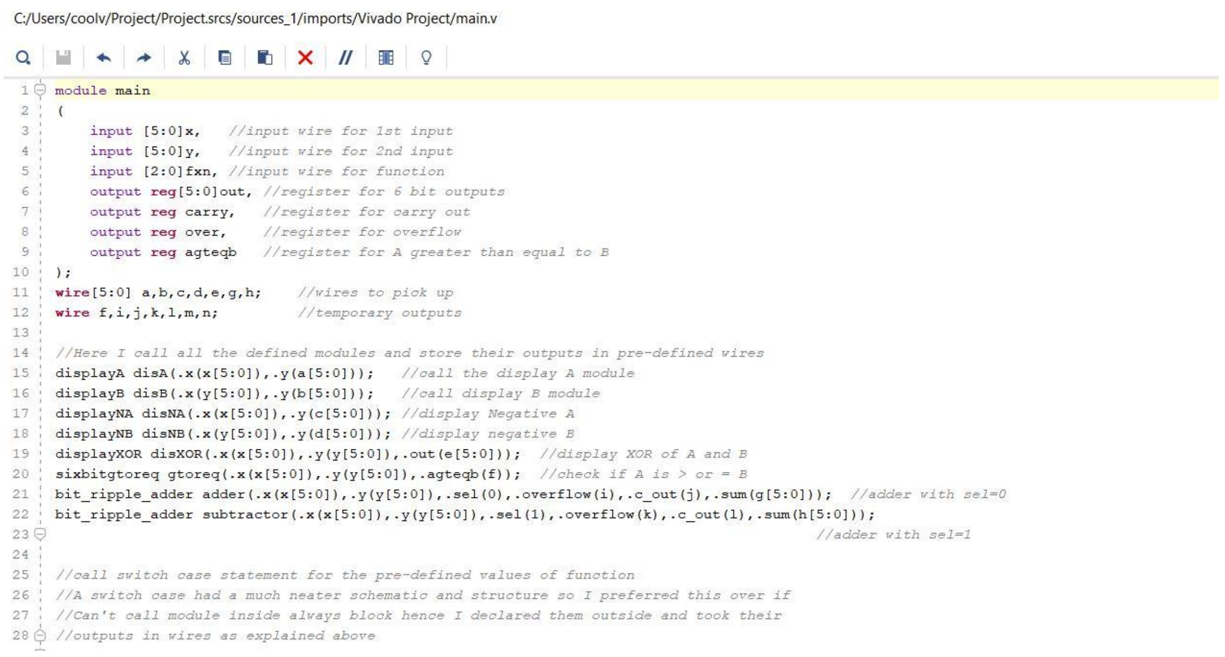
his PC > Windows (C:) > Users > coolv > Project > Project.srcs > constrs_1 > imports > Vivado Project

Name	Date modified	Type	Size
Basys3_Master.xdc	28-02-2019 17:56	XDC File	13 KB

FIGURE 3.5: Expanded Form of Constraint Sources

SECTION 4: CODE FOR MODULES, TESTBENCH AND XDC

MAIN.v (1)

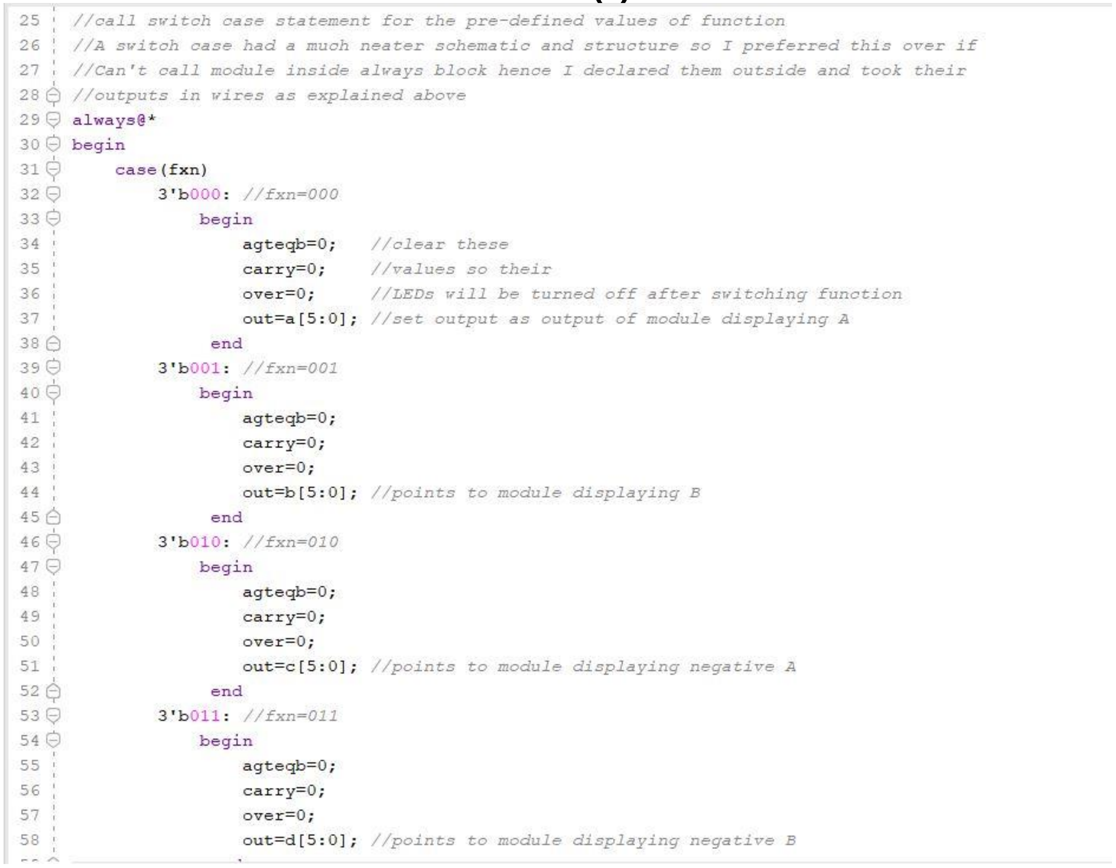


C:/Users/coolv/Project/Project.srcs/sources_1/imports/Vivado Project/main.v

```verilog
module main
(
    input [5:0]x,   //input wire for 1st input
    input [5:0]y,   //input wire for 2nd input
    input [2:0]fxn, //input wire for function
    output reg[5:0]out, //register for 6 bit outputs
    output reg carry,   //register for carry out
    output reg over,    //register for overflow
    output reg agteqb   //register for A greater than equal to B
);
wire[5:0] a,b,c,d,e,g,h;    //wires to pick up
wire f,i,j,k,l,m,n;         //temporary outputs

//Here I call all the defined modules and store their outputs in pre-defined wires
displayA disA(.x(x[5:0]),.y(a[5:0]));   //call the display A module
displayB disB(.x(y[5:0]),.y(b[5:0]));   //call display B module
displayNA disNA(.x(x[5:0]),.y(c[5:0])); //display Negative A
displayNB disNB(.x(y[5:0]),.y(d[5:0])); //display negative B
displayXOR disXOR(.x(x[5:0]),.y(y[5:0]),.out(e[5:0]));  //display XOR of A and B
sixbitgtoreq gtoreq(.x(x[5:0]),.y(y[5:0]),.agteqb(f));  //check if A is > or = B
bit_ripple_adder adder(.x(x[5:0]),.y(y[5:0]),.sel(0),.overflow(i),.c_out(j),.sum(g[5:0]));  //adder with sel=0
bit_ripple_adder subtractor(.x(x[5:0]),.y(y[5:0]),.sel(1),.overflow(k),.c_out(l),.sum(h[5:0]));
                                                                                             //adder with sel=1

//call switch case statement for the pre-defined values of function
//A switch case had a much neater schematic and structure so I preferred this over if
//Can't call module inside always block hence I declared them outside and took their
//outputs in wires as explained above
```

FIGURE 3.1: Declaring wires, registers and initializing modules

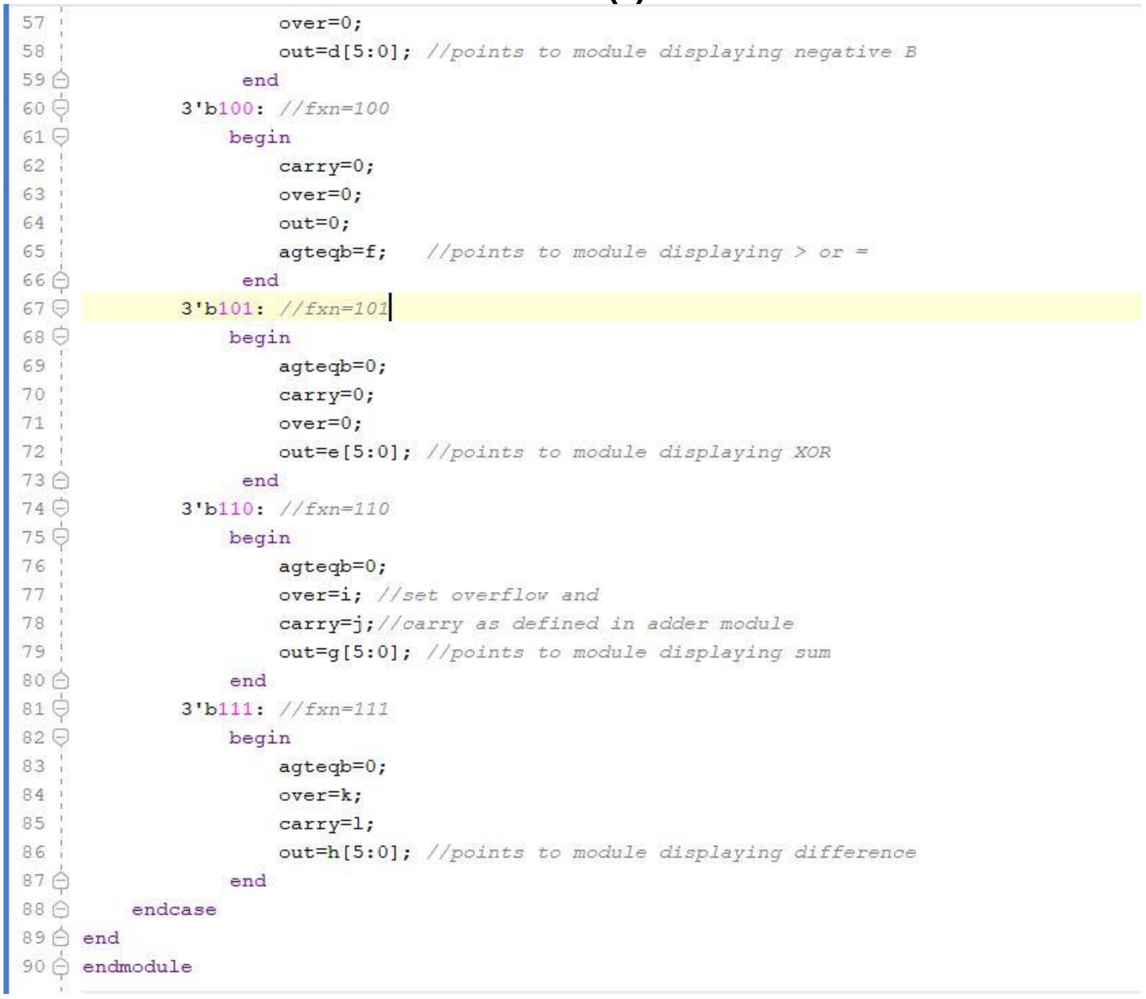
MAIN.v(2)



```verilog
//call switch case statement for the pre-defined values of function
//A switch case had a much neater schematic and structure so I preferred this over if
//Can't call module inside always block hence I declared them outside and took their
//outputs in wires as explained above
always@*
begin
    case(fxn)
        3'b000: //fxn=000
            begin
                agteqb=0;   //clear these
                carry=0;    //values so their
                over=0;     //LEDs will be turned off after switching function
                out=a[5:0]; //set output as output of module displaying A
             end
        3'b001: //fxn=001
            begin
                agteqb=0;
                carry=0;
                over=0;
                out=b[5:0]; //points to module displaying B
             end
        3'b010: //fxn=010
            begin
                agteqb=0;
                carry=0;
                over=0;
                out=c[5:0]; //points to module displaying negative A
             end
        3'b011: //fxn=011
            begin
                agteqb=0;
                carry=0;
                over=0;
                out=d[5:0]; //points to module displaying negative B
```

FIGURE 3.2: Setting the outputs of first 4 functions

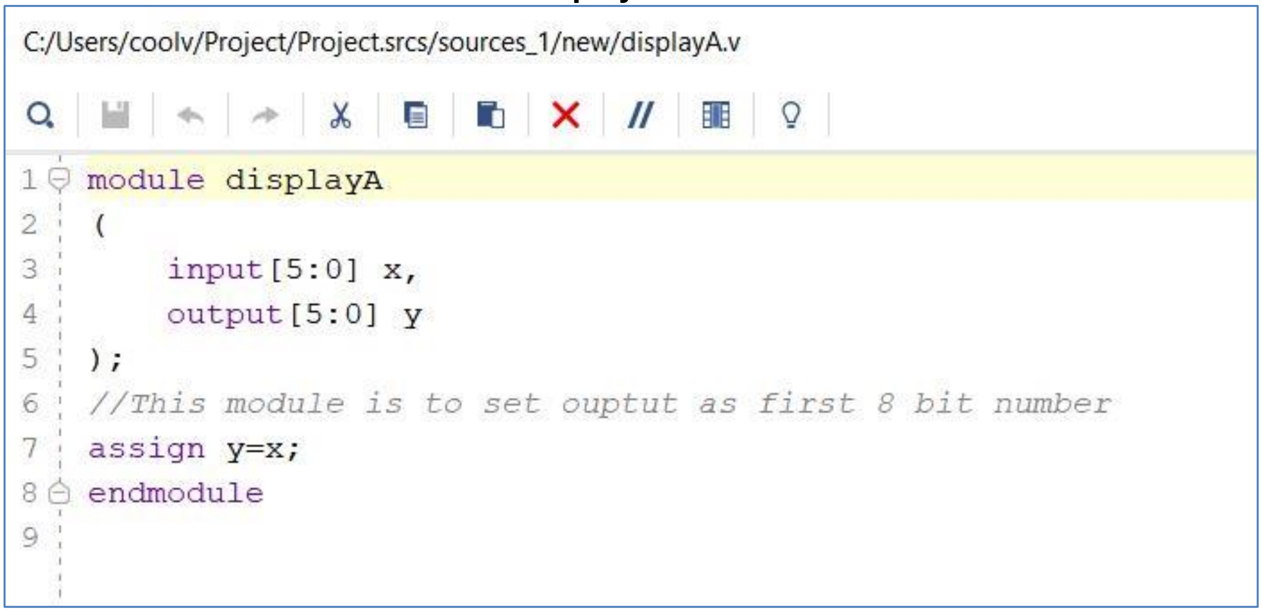
MAIN.v(3)



```
                        over=0;
                        out=d[5:0]; //points to module displaying negative B
                     end
                3'b100: //fxn=100
                    begin
                        carry=0;
                        over=0;
                        out=0;
                        agteqb=f;    //points to module displaying > or =
                     end
                3'b101: //fxn=101
                    begin
                        agteqb=0;
                        carry=0;
                        over=0;
                        out=e[5:0]; //points to module displaying XOR
                     end
                3'b110: //fxn=110
                    begin
                        agteqb=0;
                        over=i; //set overflow and
                        carry=j;//carry as defined in adder module
                        out=g[5:0]; //points to module displaying sum
                    end
                3'b111: //fxn=111
                    begin
                        agteqb=0;
                        over=k;
                        carry=l;
                        out=h[5:0]; //points to module displaying difference
                    end
        endcase
end
endmodule
```

FIGURE 3.3: Declaring outputs of next 4 function values

DisplayA.v

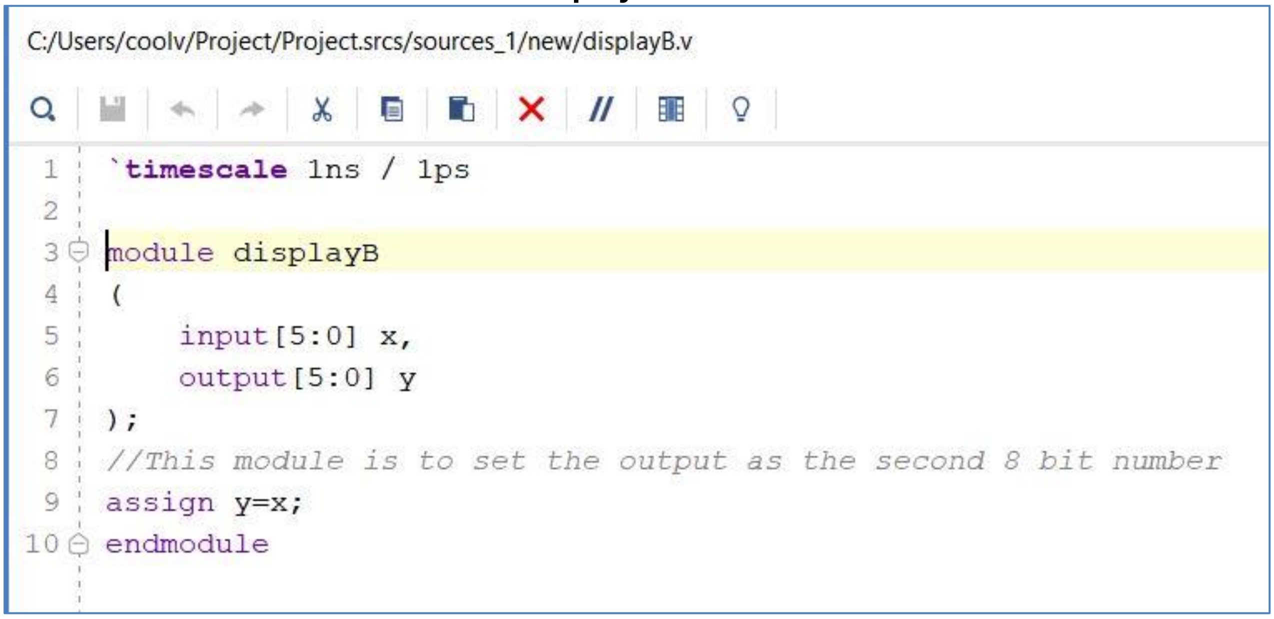


C:/Users/coolv/Project/Project.srcs/sources_1/new/displayA.v

```
module displayA
(
    input[5:0] x,
    output[5:0] y
);
//This module is to set ouptut as first 8 bit number
assign y=x;
endmodule
```

FIGURE 3.4: Module to display first input

DisplayB.v

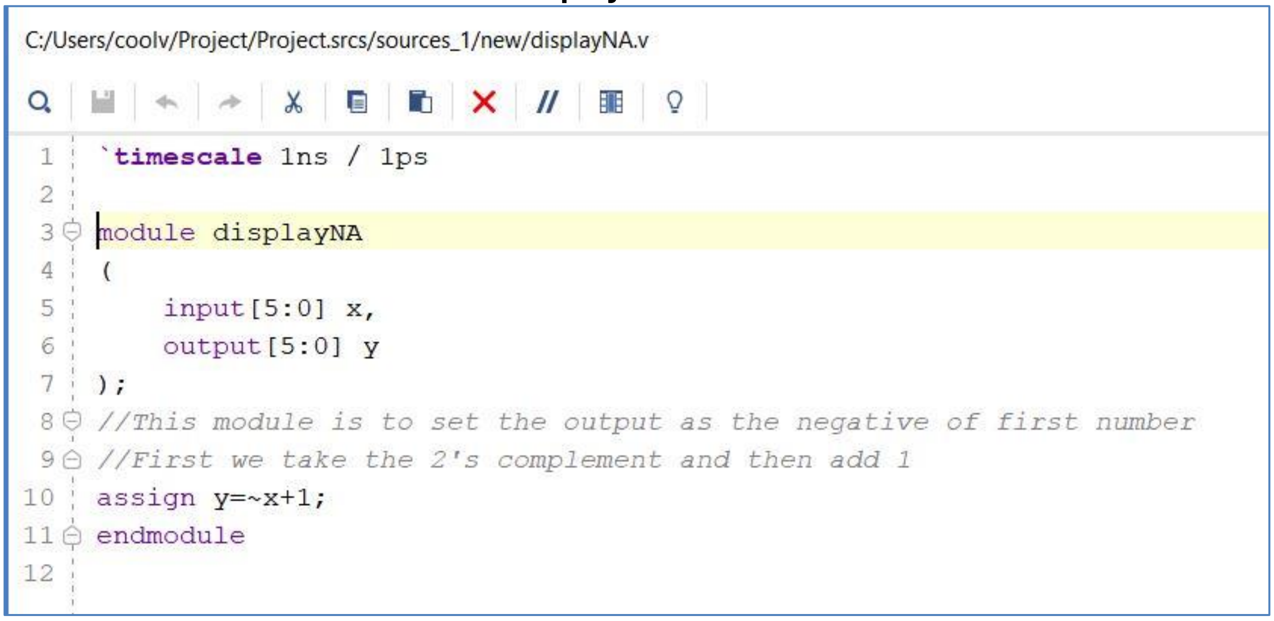


C:/Users/coolv/Project/Project.srcs/sources_1/new/displayB.v

```
`timescale 1ns / 1ps

module displayB
(
    input[5:0] x,
    output[5:0] y
);
//This module is to set the output as the second 8 bit number
assign y=x;
endmodule
```

FIGURE 3.5: Module to display second input

DisplayNA.v

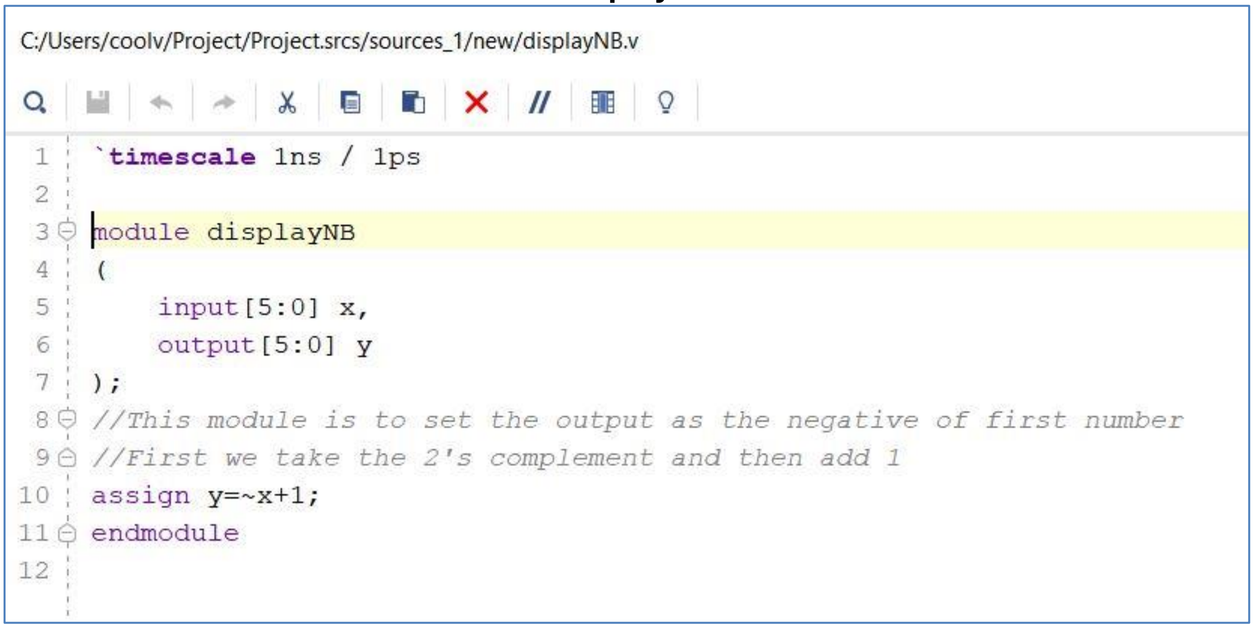


C:/Users/coolv/Project/Project.srcs/sources_1/new/displayNA.v

```
`timescale 1ns / 1ps

module displayNA
(
    input[5:0] x,
    output[5:0] y
);
//This module is to set the output as the negative of first number
//First we take the 2's complement and then add 1
assign y=~x+1;
endmodule

```

FIGURE 3.6: Display negative of first input

DisplayNB.v

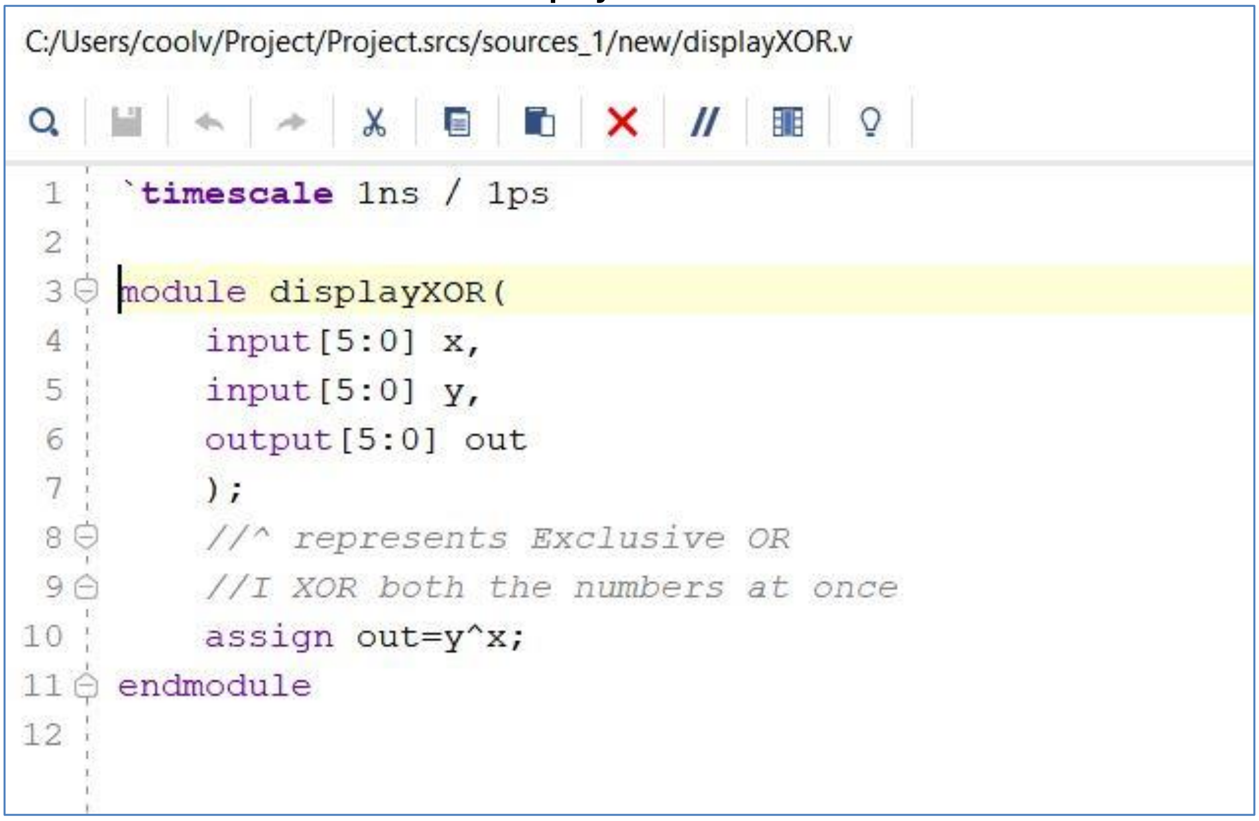


C:/Users/coolv/Project/Project.srcs/sources_1/new/displayNB.v

```
`timescale 1ns / 1ps

module displayNB
(
    input[5:0] x,
    output[5:0] y
);
//This module is to set the output as the negative of first number
//First we take the 2's complement and then add 1
assign y=~x+1;
endmodule

```

FIGURE 3.7: Display negative of second input

DisplayXOR.v

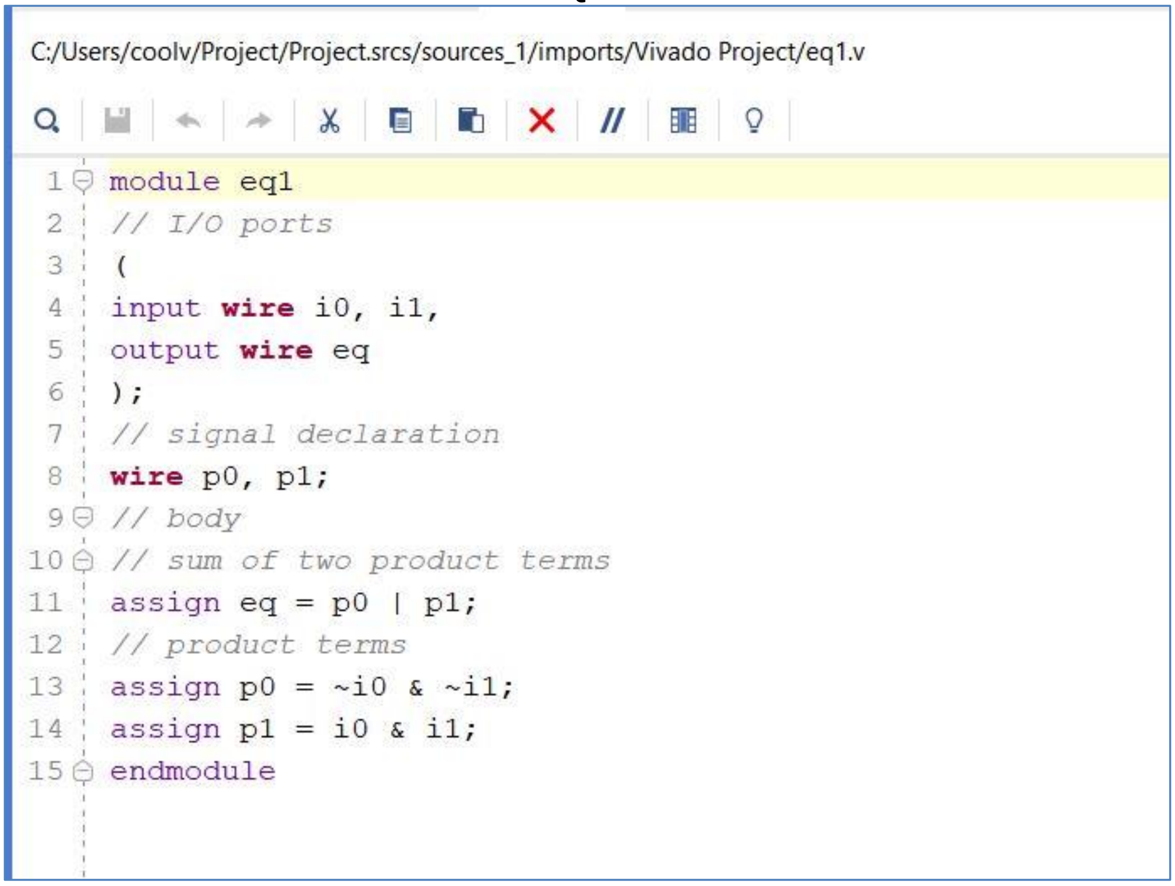


C:/Users/coolv/Project/Project.srcs/sources_1/new/displayXOR.v

```
`timescale 1ns / 1ps

module displayXOR(
    input[5:0] x,
    input[5:0] y,
    output[5:0] out
    );
    //^ represents Exclusive OR
    //I XOR both the numbers at once
    assign out=y^x;
endmodule

```

FIGURE 3.8: Display XOR of both inputs

EQ1.v

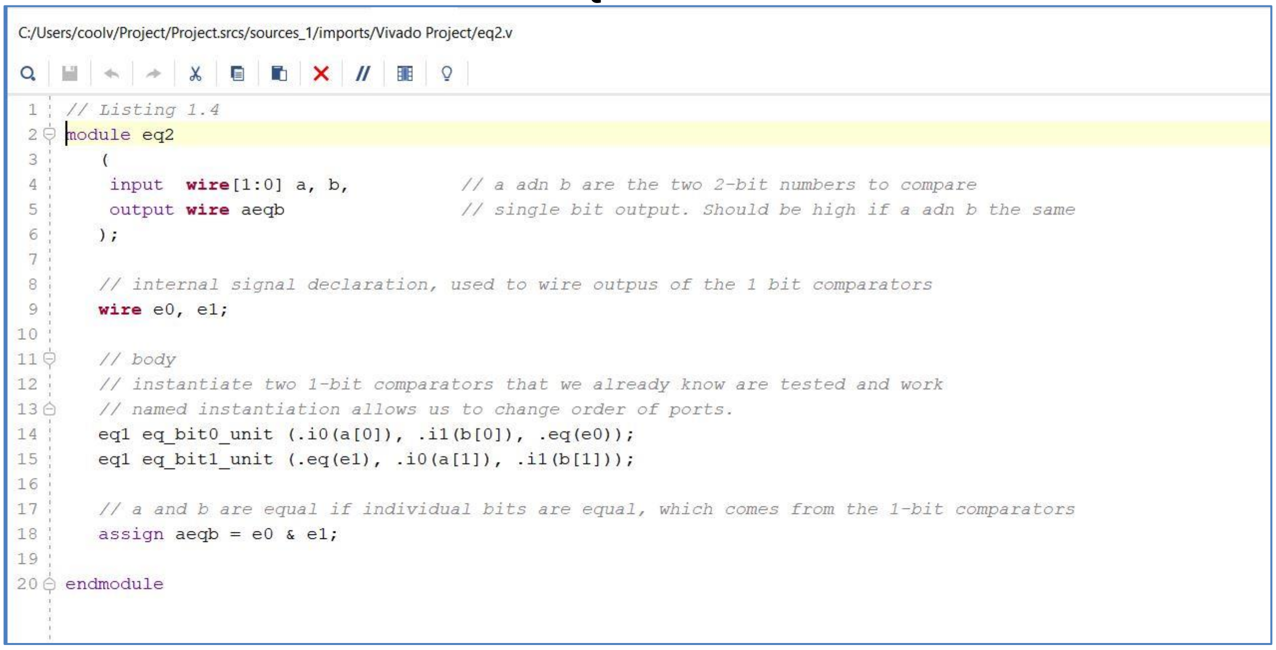


C:/Users/coolv/Project/Project.srcs/sources_1/imports/Vivado Project/eq1.v

```verilog
module eq1
// I/O ports
(
input wire i0, i1,
output wire eq
);
// signal declaration
wire p0, p1;
// body
// sum of two product terms
assign eq = p0 | p1;
// product terms
assign p0 = ~i0 & ~i1;
assign p1 = i0 & i1;
endmodule
```

FIGURE 3.9: 1-bit equal to module

EQ2.v

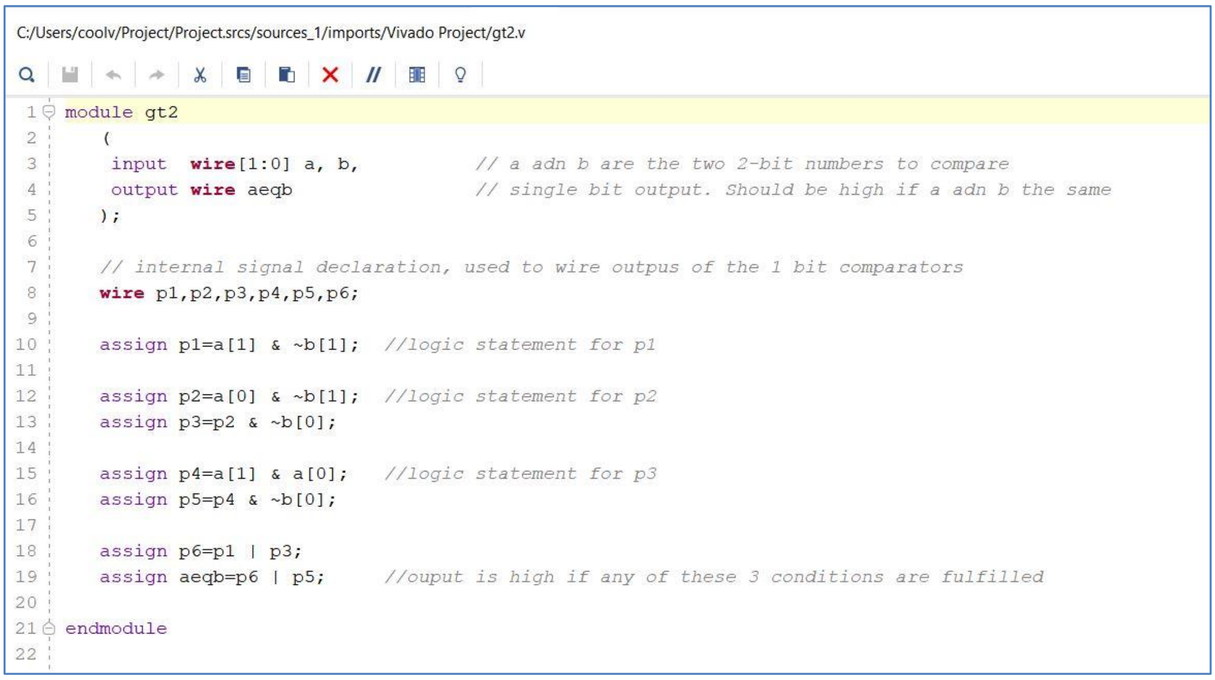


C:/Users/coolv/Project/Project.srcs/sources_1/imports/Vivado Project/eq2.v

```verilog
// Listing 1.4
module eq2
    (
     input  wire[1:0] a, b,          // a adn b are the two 2-bit numbers to compare
     output wire aeqb                // single bit output. Should be high if a adn b the same
    );

    // internal signal declaration, used to wire outpus of the 1 bit comparators
    wire e0, e1;

    // body
    // instantiate two 1-bit comparators that we already know are tested and work
    // named instantiation allows us to change order of ports.
    eq1 eq_bit0_unit (.i0(a[0]), .i1(b[0]), .eq(e0));
    eq1 eq_bit1_unit (.eq(e1), .i0(a[1]), .i1(b[1]));

    // a and b are equal if individual bits are equal, which comes from the 1-bit comparators
    assign aeqb = e0 & e1;

endmodule
```

FIGURE 3.10: 2-bit equal to module

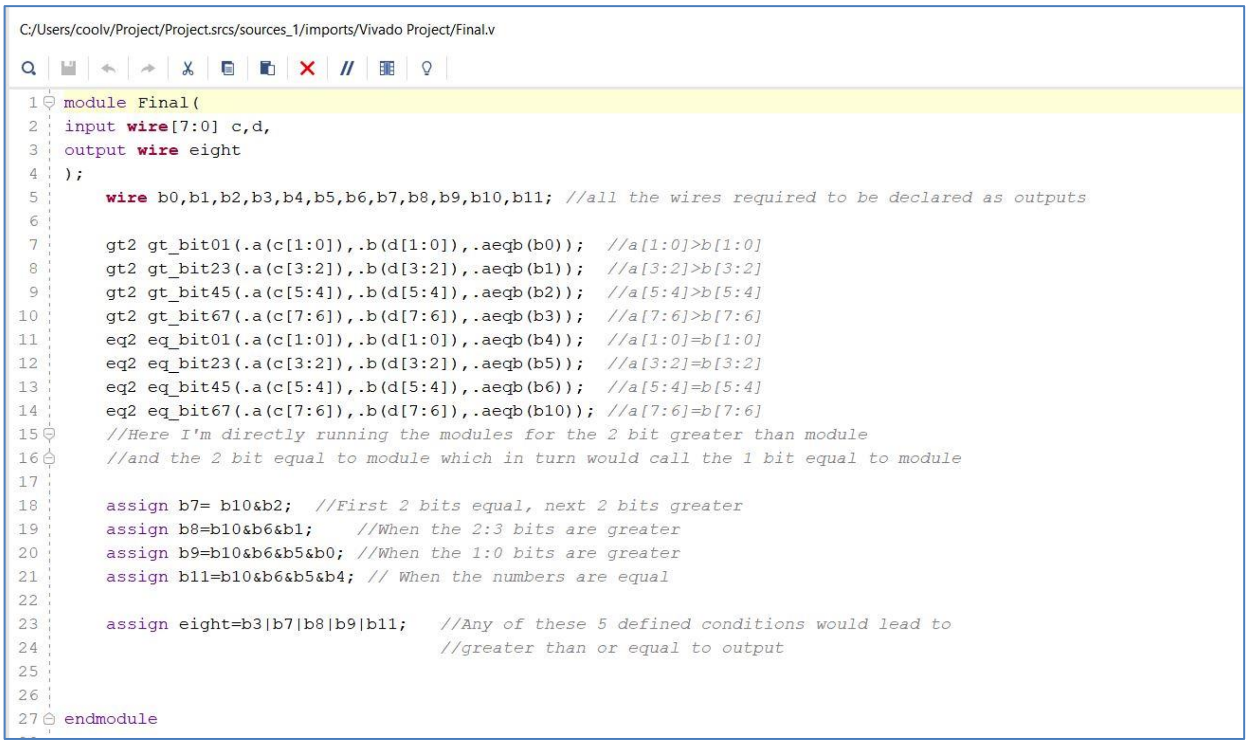
GT2.v


C:/Users/coolv/Project/Project.srcs/sources_1/imports/Vivado Project/gt2.v

```
module gt2
    (
     input  wire[1:0] a, b,          // a adn b are the two 2-bit numbers to compare
     output wire aeqb                // single bit output. Should be high if a adn b the same
    );

    // internal signal declaration, used to wire outpus of the 1 bit comparators
    wire p1,p2,p3,p4,p5,p6;

    assign p1=a[1] & ~b[1];  //logic statement for p1

    assign p2=a[0] & ~b[1];  //logic statement for p2
    assign p3=p2 & ~b[0];

    assign p4=a[1] & a[0];   //logic statement for p3
    assign p5=p4 & ~b[0];

    assign p6=p1 | p3;
    assign aeqb=p6 | p5;     //ouput is high if any of these 3 conditions are fulfilled

endmodule

```

FIGURE 3.11: 2-bit greater than module

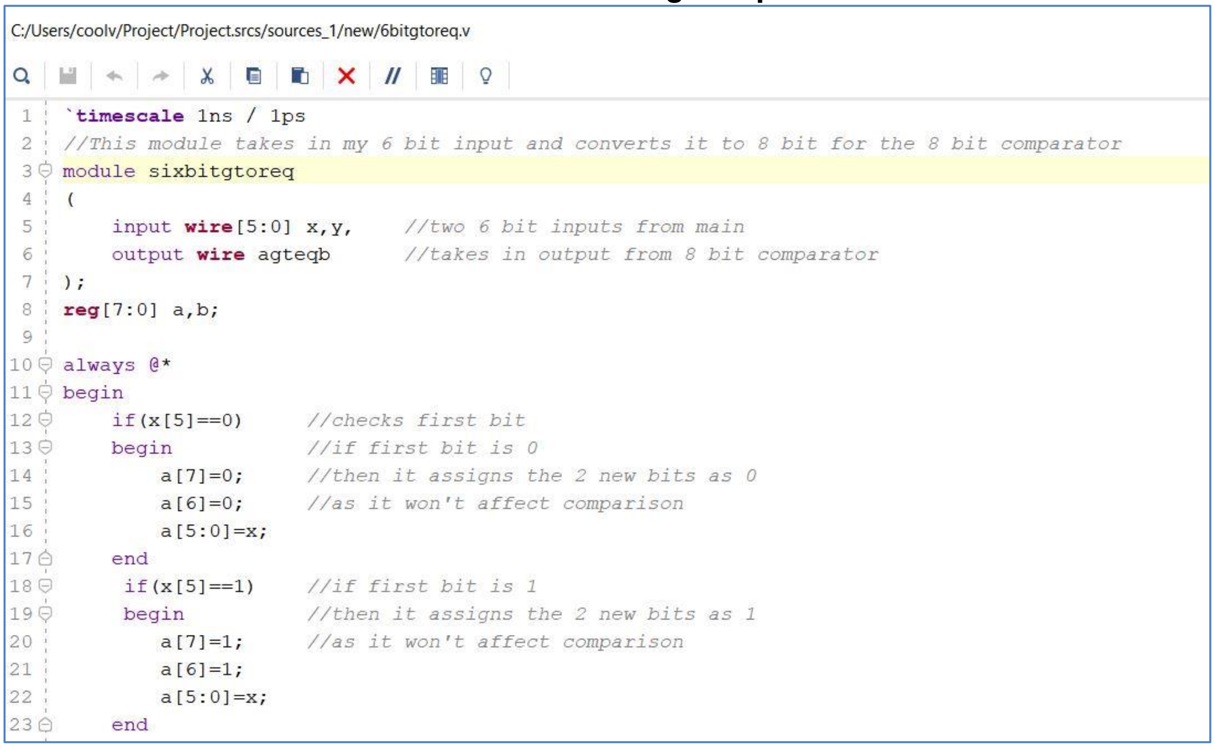
Final.v


C:/Users/coolv/Project/Project.srcs/sources_1/imports/Vivado Project/Final.v

```
module Final(
input wire[7:0] c,d,
output wire eight
);
    wire b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11; //all the wires required to be declared as outputs

    gt2 gt_bit01(.a(c[1:0]),.b(d[1:0]),.aeqb(b0));  //a[1:0]>b[1:0]
    gt2 gt_bit23(.a(c[3:2]),.b(d[3:2]),.aeqb(b1));  //a[3:2]>b[3:2]
    gt2 gt_bit45(.a(c[5:4]),.b(d[5:4]),.aeqb(b2));  //a[5:4]>b[5:4]
    gt2 gt_bit67(.a(c[7:6]),.b(d[7:6]),.aeqb(b3));  //a[7:6]>b[7:6]
    eq2 eq_bit01(.a(c[1:0]),.b(d[1:0]),.aeqb(b4));  //a[1:0]=b[1:0]
    eq2 eq_bit23(.a(c[3:2]),.b(d[3:2]),.aeqb(b5));  //a[3:2]=b[3:2]
    eq2 eq_bit45(.a(c[5:4]),.b(d[5:4]),.aeqb(b6));  //a[5:4]=b[5:4]
    eq2 eq_bit67(.a(c[7:6]),.b(d[7:6]),.aeqb(b10)); //a[7:6]=b[7:6]
    //Here I'm directly running the modules for the 2 bit greater than module
    //and the 2 bit equal to module which in turn would call the 1 bit equal to module

    assign b7= b10&b2;  //First 2 bits equal, next 2 bits greater
    assign b8=b10&b6&b1;    //When the 2:3 bits are greater
    assign b9=b10&b6&b5&b0; //When the 1:0 bits are greater
    assign b11=b10&b6&b5&b4; // When the numbers are equal

    assign eight=b3|b7|b8|b9|b11;   //Any of these 5 defined conditions would lead to
                                    //greater than or equal to output


endmodule
```

FIGURE 3.12: 8-bit greater than equal to module

6bitgtoreq.v

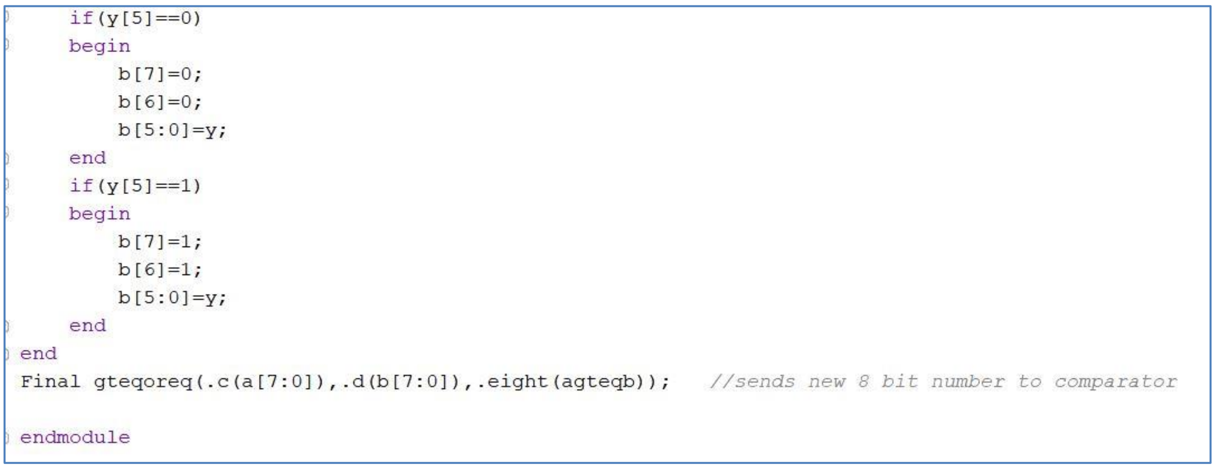


C:/Users/coolv/Project/Project.srcs/sources_1/new/6bitgtoreq.v

```verilog
`timescale 1ns / 1ps
//This module takes in my 6 bit input and converts it to 8 bit for the 8 bit comparator
module sixbitgtoreq
(
    input wire[5:0] x,y,     //two 6 bit inputs from main
    output wire agteqb       //takes in output from 8 bit comparator
);
reg[7:0] a,b;

always @*
begin
    if(x[5]==0)      //checks first bit
    begin            //if first bit is 0
        a[7]=0;      //then it assigns the 2 new bits as 0
        a[6]=0;      //as it won't affect comparison
        a[5:0]=x;
    end
     if(x[5]==1)     //if first bit is 1
     begin           //then it assigns the 2 new bits as 1
        a[7]=1;      //as it won't affect comparison
        a[6]=1;
        a[5:0]=x;
    end
```

FIGURE 3.13: 6-bit greater than or equal to module

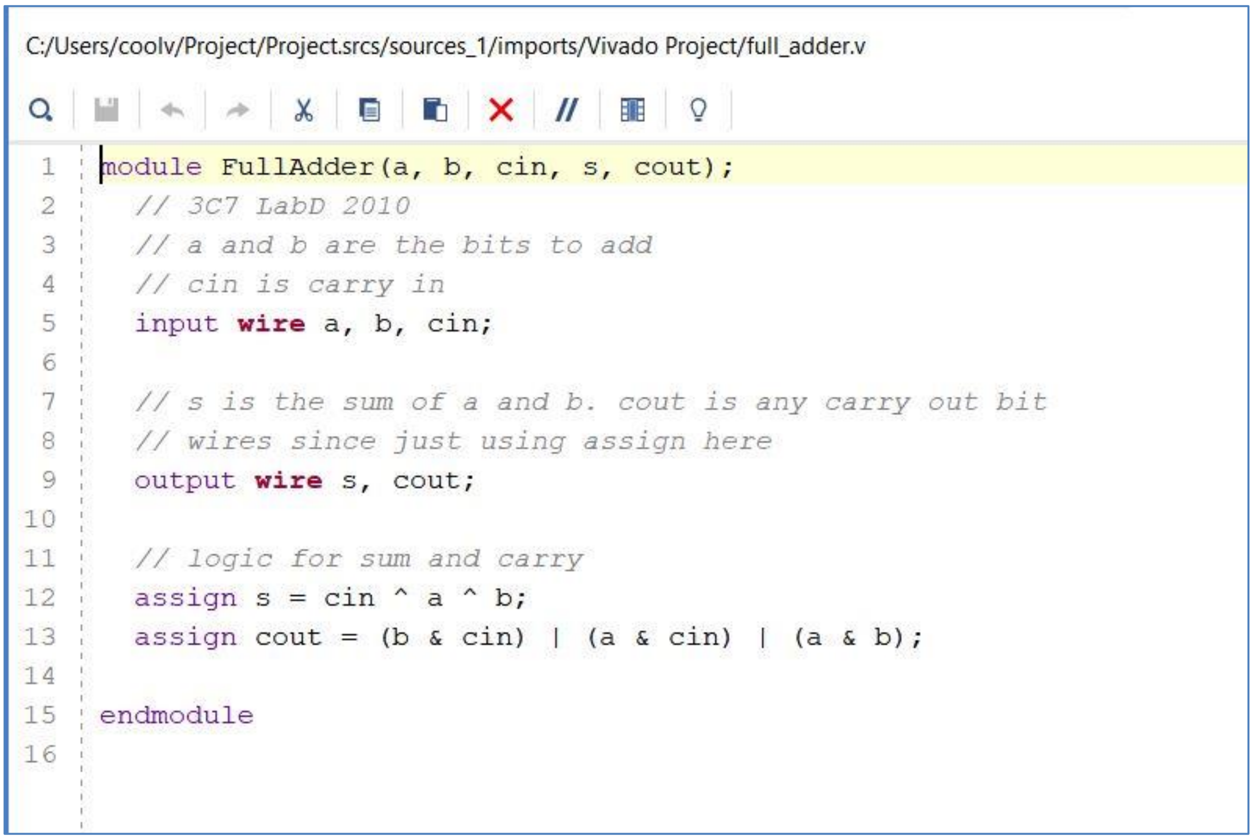
6bitgtoreq.v(2)



```verilog
    if(y[5]==0)
    begin
        b[7]=0;
        b[6]=0;
        b[5:0]=y;
    end
    if(y[5]==1)
    begin
        b[7]=1;
        b[6]=1;
        b[5:0]=y;
    end
end
Final gteqoreq(.c(a[7:0]),.d(b[7:0]),.eight(agteqb));   //sends new 8 bit number to comparator

endmodule
```

FIGURE 3.14: 6-bit greater than or equal to module

Full_adder.v

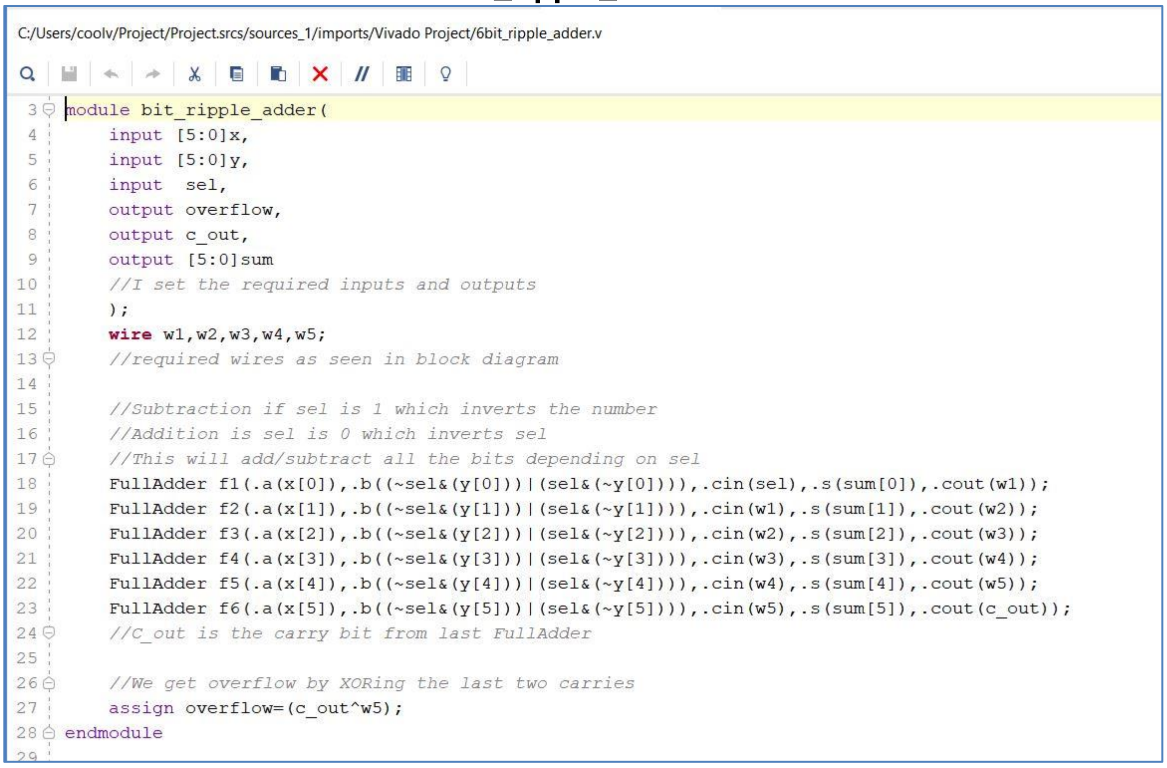


C:/Users/coolv/Project/Project.srcs/sources_1/imports/Vivado Project/full_adder.v

```verilog
module FullAdder(a, b, cin, s, cout);
  // 3C7 LabD 2010
  // a and b are the bits to add
  // cin is carry in
  input wire a, b, cin;

  // s is the sum of a and b. cout is any carry out bit
  // wires since just using assign here
  output wire s, cout;

  // logic for sum and carry
  assign s = cin ^ a ^ b;
  assign cout = (b & cin) | (a & cin) | (a & b);

endmodule

```

FIGURE 3.15: 2 bit Full Adder Module

6bit_ripple_adder.v

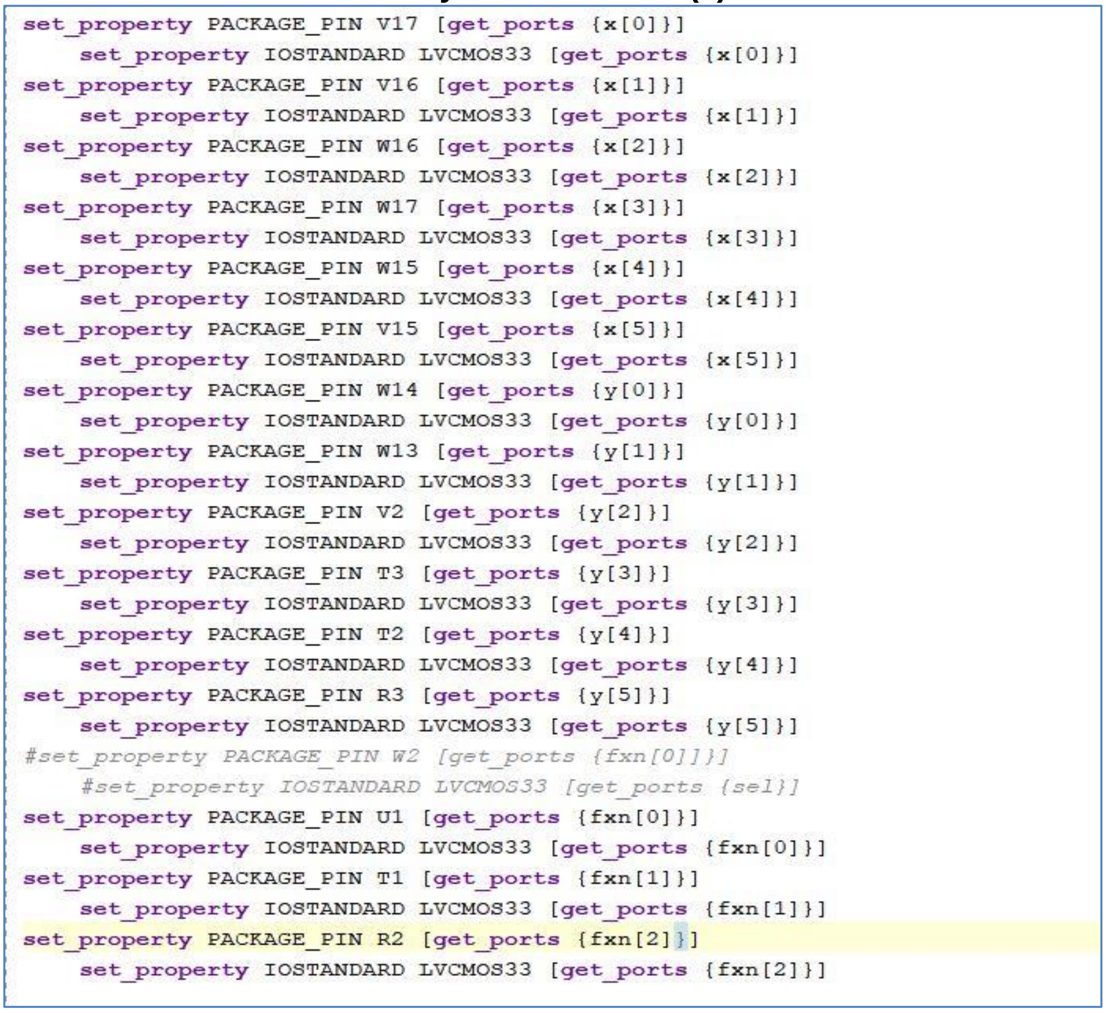


C:/Users/coolv/Project/Project.srcs/sources_1/imports/Vivado Project/6bit_ripple_adder.v

```verilog
module bit_ripple_adder(
    input [5:0]x,
    input [5:0]y,
    input  sel,
    output overflow,
    output c_out,
    output [5:0]sum
    //I set the required inputs and outputs
    );
    wire w1,w2,w3,w4,w5;
    //required wires as seen in block diagram

    //Subtraction if sel is 1 which inverts the number
    //Addition is sel is 0 which inverts sel
    //This will add/subtract all the bits depending on sel
    FullAdder f1(.a(x[0]),.b((~sel&(y[0]))|(sel&(~y[0]))),.cin(sel),.s(sum[0]),.cout(w1));
    FullAdder f2(.a(x[1]),.b((~sel&(y[1]))|(sel&(~y[1]))),.cin(w1),.s(sum[1]),.cout(w2));
    FullAdder f3(.a(x[2]),.b((~sel&(y[2]))|(sel&(~y[2]))),.cin(w2),.s(sum[2]),.cout(w3));
    FullAdder f4(.a(x[3]),.b((~sel&(y[3]))|(sel&(~y[3]))),.cin(w3),.s(sum[3]),.cout(w4));
    FullAdder f5(.a(x[4]),.b((~sel&(y[4]))|(sel&(~y[4]))),.cin(w4),.s(sum[4]),.cout(w5));
    FullAdder f6(.a(x[5]),.b((~sel&(y[5]))|(sel&(~y[5]))),.cin(w5),.s(sum[5]),.cout(c_out));
    //C_out is the carry bit from last FullAdder

    //We get overflow by XORing the last two carries
    assign overflow=(c_out^w5);
endmodule
```

FIGURE 3.16: 6-bit adder/subtractor module

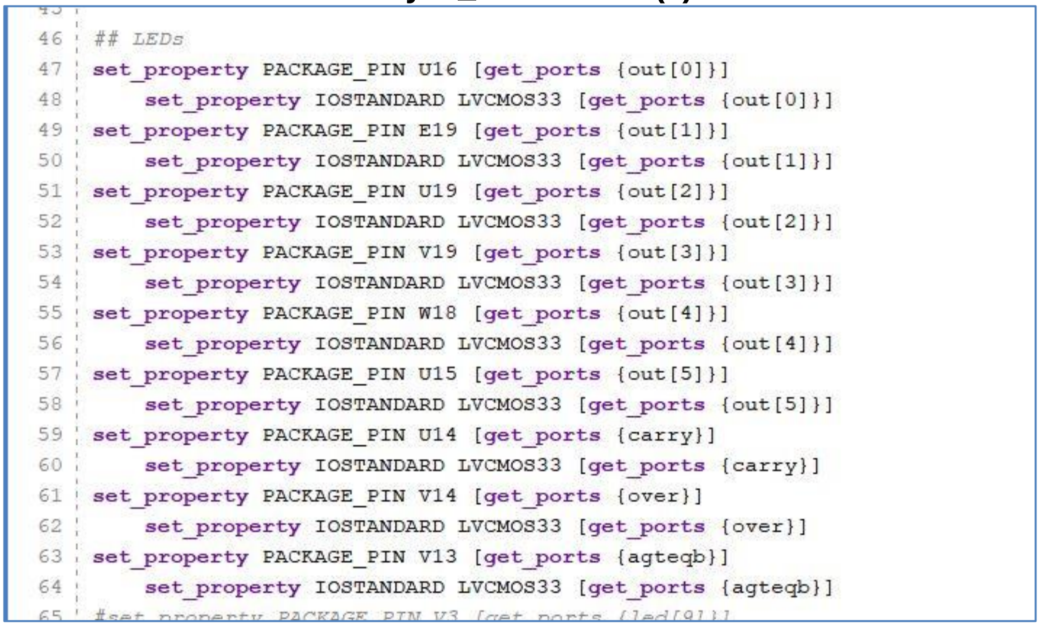
Basys3_master.xdc (1)



```
set_property PACKAGE_PIN V17 [get_ports {x[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {x[0]}]
set_property PACKAGE_PIN V16 [get_ports {x[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {x[1]}]
set_property PACKAGE_PIN W16 [get_ports {x[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {x[2]}]
set_property PACKAGE_PIN W17 [get_ports {x[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {x[3]}]
set_property PACKAGE_PIN W15 [get_ports {x[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {x[4]}]
set_property PACKAGE_PIN V15 [get_ports {x[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {x[5]}]
set_property PACKAGE_PIN W14 [get_ports {y[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {y[0]}]
set_property PACKAGE_PIN W13 [get_ports {y[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {y[1]}]
set_property PACKAGE_PIN V2 [get_ports {y[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {y[2]}]
set_property PACKAGE_PIN T3 [get_ports {y[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {y[3]}]
set_property PACKAGE_PIN T2 [get_ports {y[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {y[4]}]
set_property PACKAGE_PIN R3 [get_ports {y[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {y[5]}]
#set_property PACKAGE_PIN W2 [get_ports {fxn[0]}]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sel}]
set_property PACKAGE_PIN U1 [get_ports {fxn[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {fxn[0]}]
set_property PACKAGE_PIN T1 [get_ports {fxn[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {fxn[1]}]
set_property PACKAGE_PIN R2 [get_ports {fxn[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {fxn[2]}]
```

FIGURE 3.17: Creating XDC file inputs

Basys3_master.xdc (2)



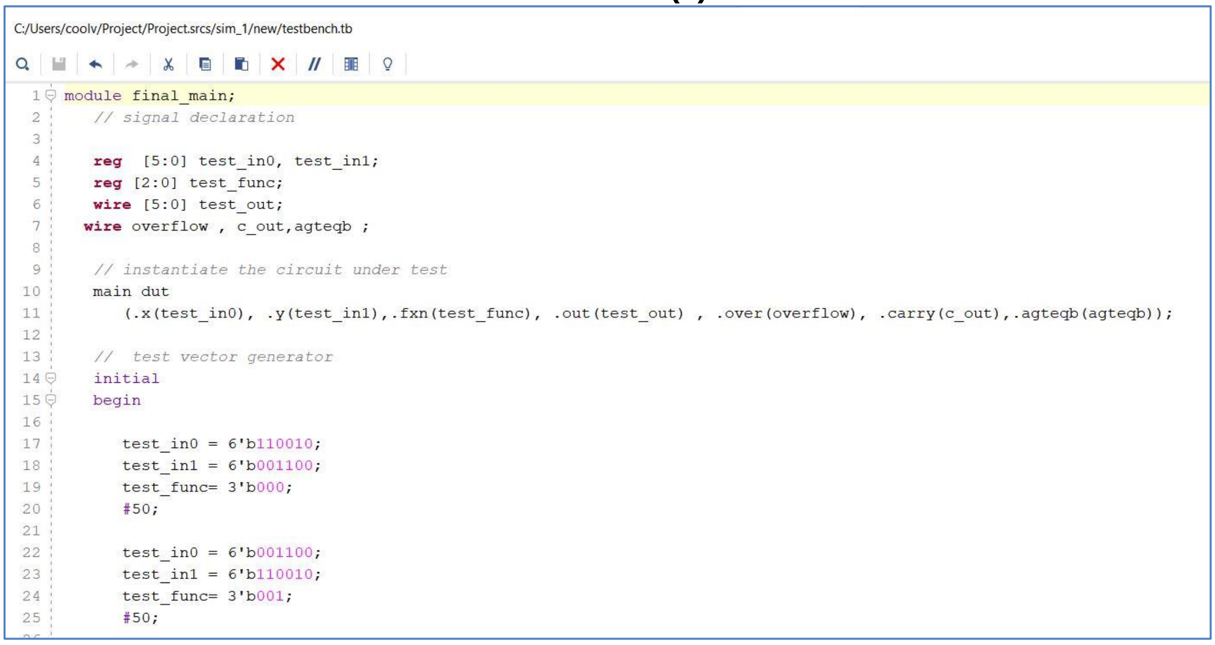
```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {out[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out[0]}]
set_property PACKAGE_PIN E19 [get_ports {out[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out[1]}]
set_property PACKAGE_PIN U19 [get_ports {out[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out[2]}]
set_property PACKAGE_PIN V19 [get_ports {out[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out[3]}]
set_property PACKAGE_PIN W18 [get_ports {out[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out[4]}]
set_property PACKAGE_PIN U15 [get_ports {out[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out[5]}]
set_property PACKAGE_PIN U14 [get_ports {carry}]
    set_property IOSTANDARD LVCMOS33 [get_ports {carry}]
set_property PACKAGE_PIN V14 [get_ports {over}]
    set_property IOSTANDARD LVCMOS33 [get_ports {over}]
set_property PACKAGE_PIN V13 [get_ports {agteqb}]
    set_property IOSTANDARD LVCMOS33 [get_ports {agteqb}]
#set_property PACKAGE_PIN V3 [get_ports {led[9]}]
```

FIGURE 3.18: Creating XDC file outputs

TESTBENCH

I’ve designed exhaustive testbenches to test all the above modules exhaustively.
For all the individual modules, I tested them far more rigorously in my previous lab sessions as you can see in my previous reports which I’ve included.
All the cases gave me outputs as expected and hence I feel that the Arithmetic Logic Unit is a success as a whole.

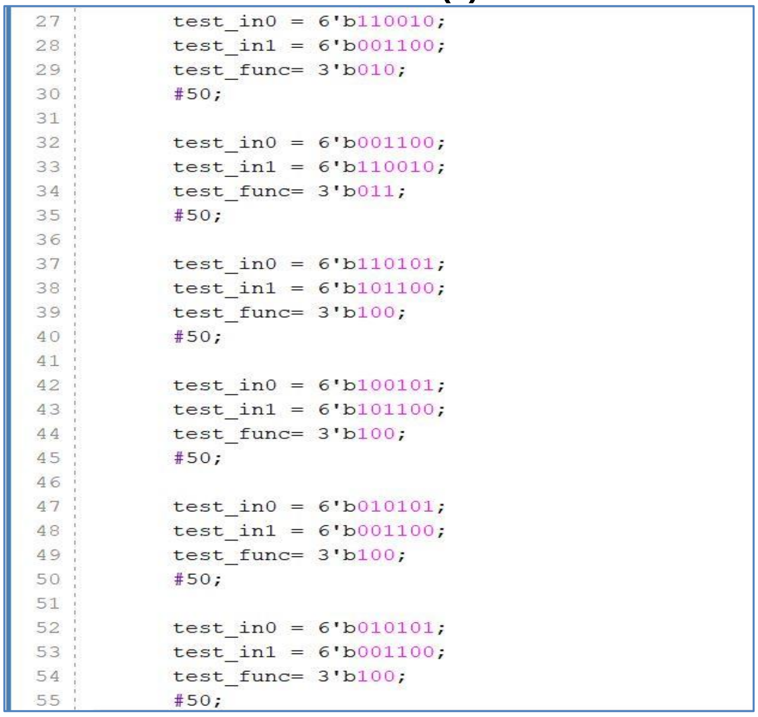
Testbench (1)



C:/Users/coolv/Project/Project.srcs/sim_1/new/testbench.tb

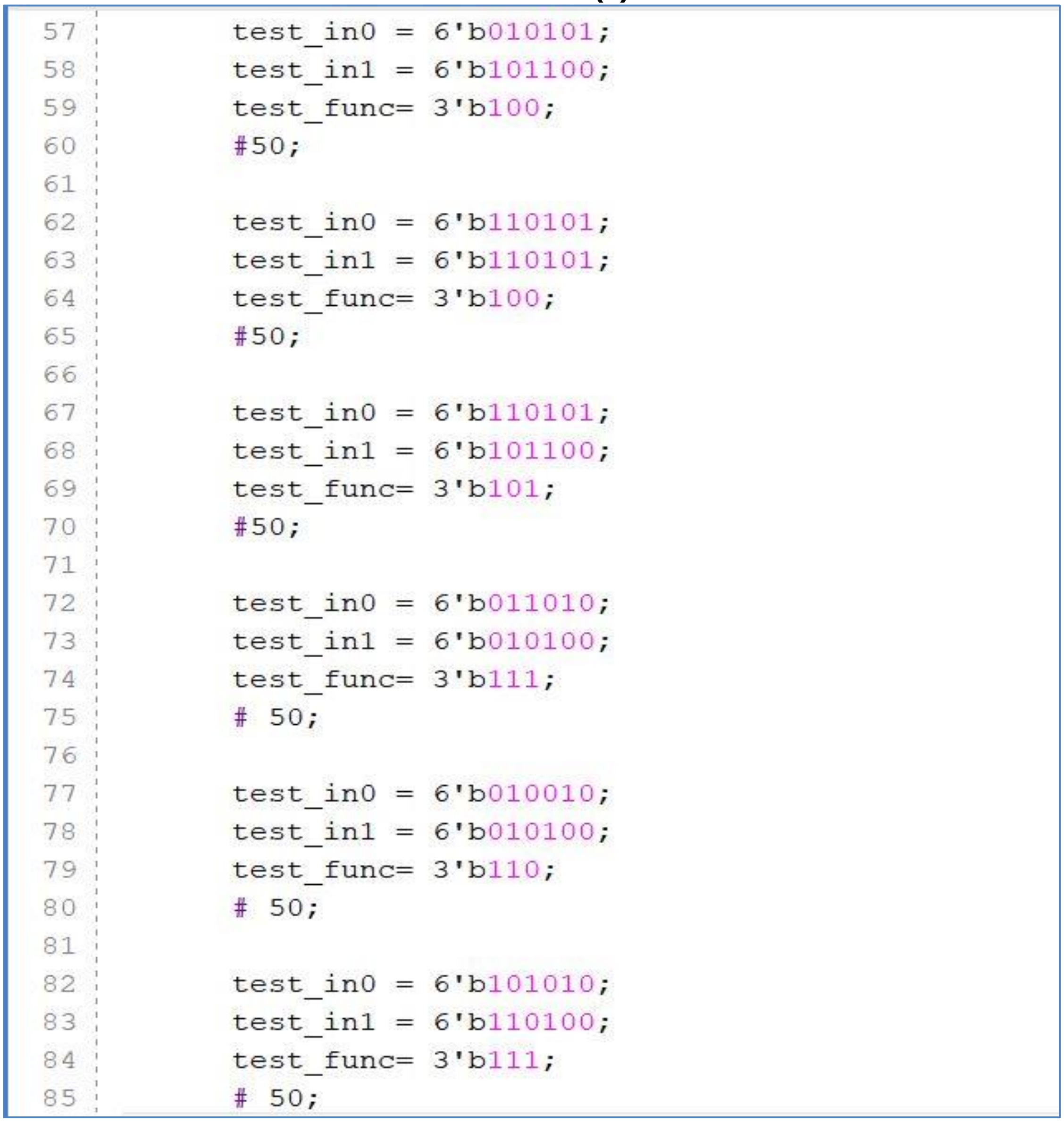
```
module final_main;
   // signal declaration

   reg  [5:0] test_in0, test_in1;
   reg [2:0] test_func;
   wire [5:0] test_out;
  wire overflow , c_out,agteqb ;

   // instantiate the circuit under test
   main dut
      (.x(test_in0), .y(test_in1),.fxn(test_func), .out(test_out) , .over(overflow), .carry(c_out),.agteqb(agteqb));

   //  test vector generator
   initial
   begin

      test_in0 = 6'b110010;
      test_in1 = 6'b001100;
      test_func= 3'b000;
      #50;

      test_in0 = 6'b001100;
      test_in1 = 6'b110010;
      test_func= 3'b001;
      #50;
```

Testbench(2)



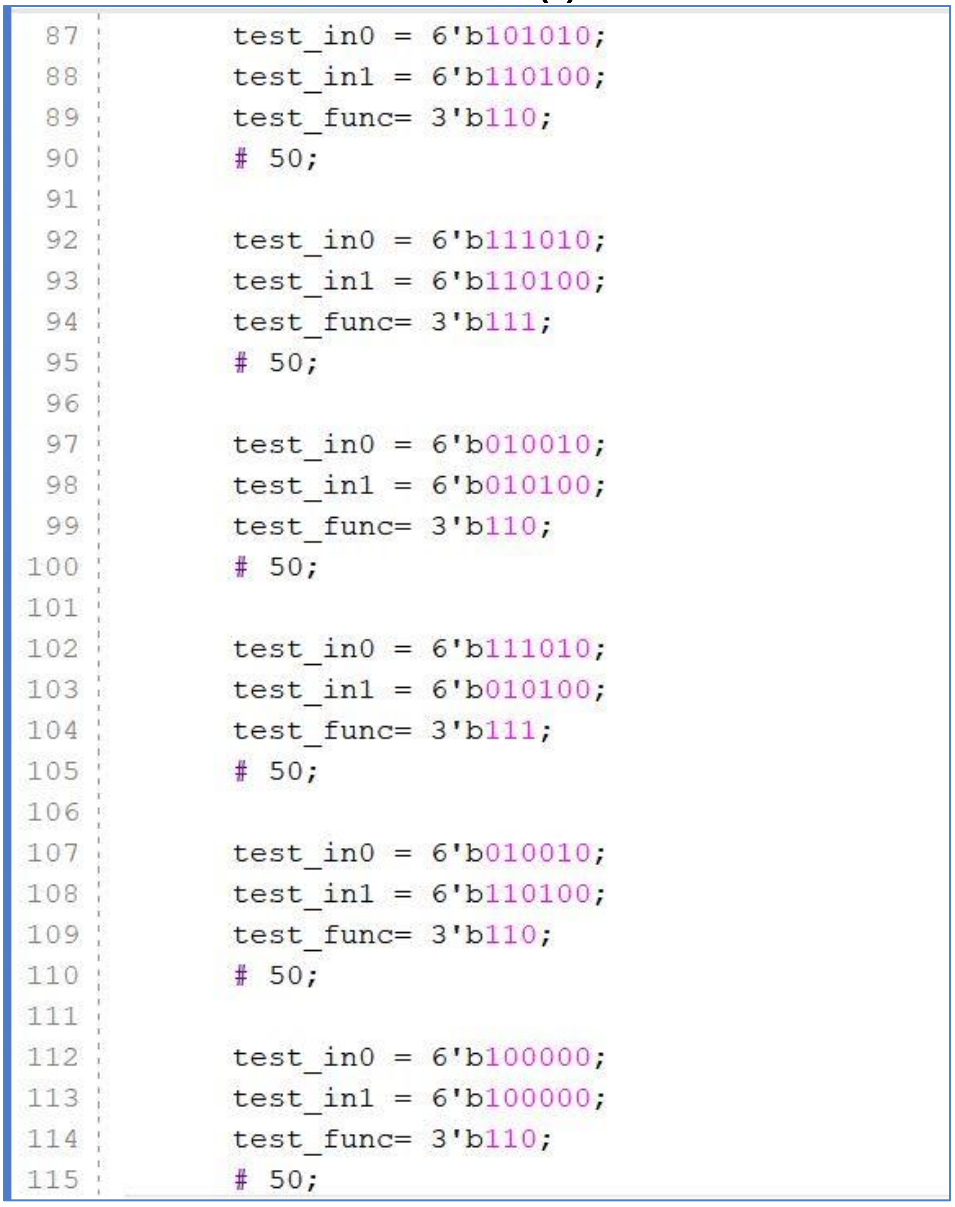
```
      test_in0 = 6'b110010;
      test_in1 = 6'b001100;
      test_func= 3'b010;
      #50;

      test_in0 = 6'b001100;
      test_in1 = 6'b110010;
      test_func= 3'b011;
      #50;

      test_in0 = 6'b110101;
      test_in1 = 6'b101100;
      test_func= 3'b100;
      #50;

      test_in0 = 6'b100101;
      test_in1 = 6'b101100;
      test_func= 3'b100;
      #50;

      test_in0 = 6'b010101;
      test_in1 = 6'b001100;
      test_func= 3'b100;
      #50;

      test_in0 = 6'b010101;
      test_in1 = 6'b001100;
      test_func= 3'b100;
      #50;
```

Testbench (3)



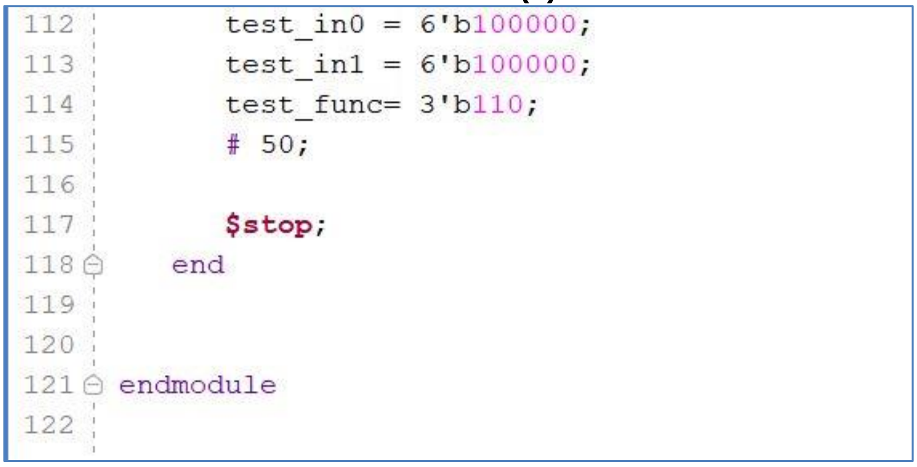
```
        test_in0 = 6'b010101;
        test_in1 = 6'b101100;
        test_func= 3'b100;
        #50;

        test_in0 = 6'b110101;
        test_in1 = 6'b110101;
        test_func= 3'b100;
        #50;

        test_in0 = 6'b110101;
        test_in1 = 6'b101100;
        test_func= 3'b101;
        #50;

        test_in0 = 6'b011010;
        test_in1 = 6'b010100;
        test_func= 3'b111;
        # 50;

        test_in0 = 6'b010010;
        test_in1 = 6'b010100;
        test_func= 3'b110;
        # 50;

        test_in0 = 6'b101010;
        test_in1 = 6'b110100;
        test_func= 3'b111;
        # 50;
```

Testbench(4)



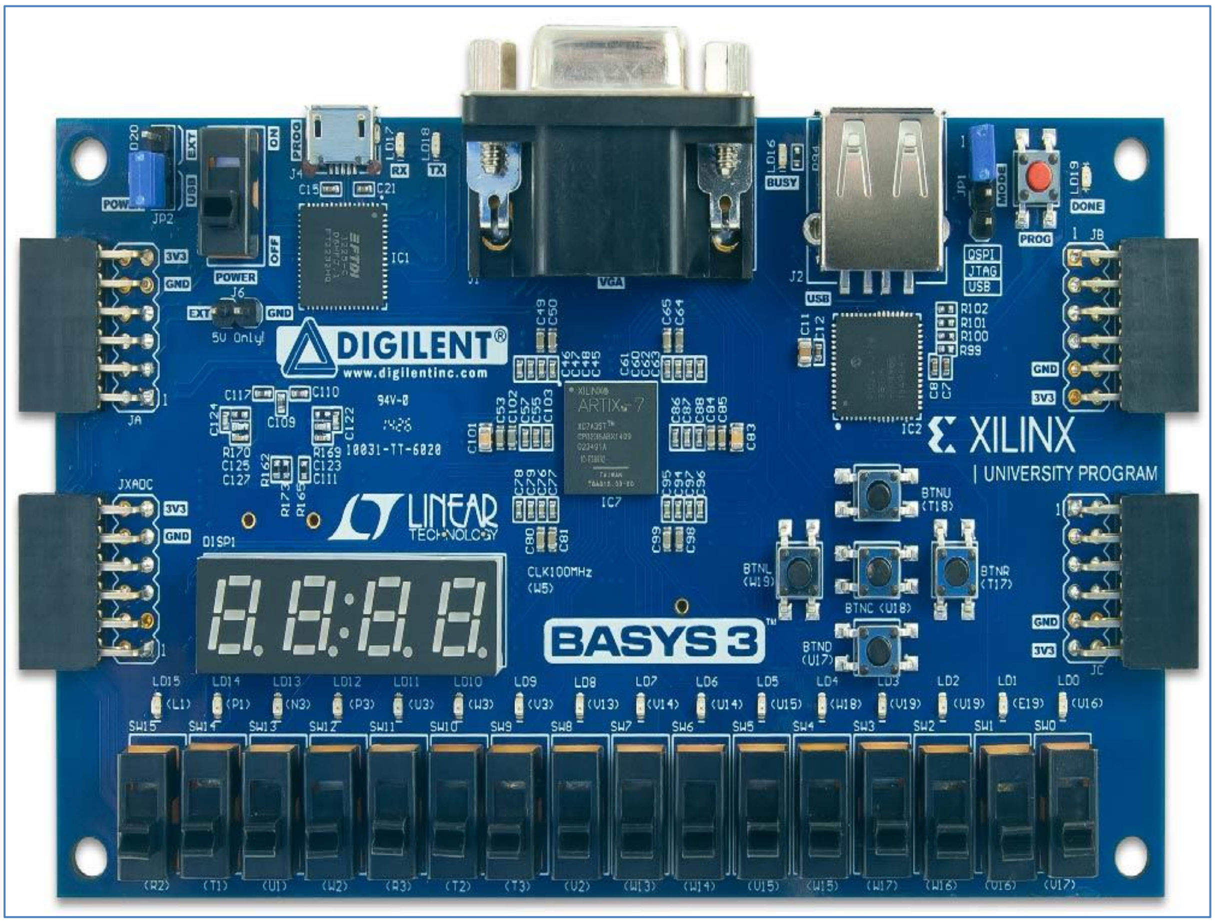
```
        test_in0 = 6'b101010;
        test_in1 = 6'b110100;
        test_func= 3'b110;
        # 50;

        test_in0 = 6'b111010;
        test_in1 = 6'b110100;
        test_func= 3'b111;
        # 50;

        test_in0 = 6'b010010;
        test_in1 = 6'b010100;
        test_func= 3'b110;
        # 50;

        test_in0 = 6'b111010;
        test_in1 = 6'b010100;
        test_func= 3'b111;
        # 50;

        test_in0 = 6'b010010;
        test_in1 = 6'b110100;
        test_func= 3'b110;
        # 50;

        test_in0 = 6'b100000;
        test_in1 = 6'b100000;
        test_func= 3'b110;
        # 50;
```

Testbench(5)



```
        test_in0 = 6'b100000;
        test_in1 = 6'b100000;
        test_func= 3'b110;
        # 50;

        $stop;
    end


endmodule

```

DEMO



INPUTS
The switches R2, T1 and V1 are assigned as the 3 bits of the function, with R2 being the most significant bit.
The 6 switches from (V15-V17) are assigned as the 6-bit input for A, with V15 being the most significant bit.
The 6 switches from (R3-W14) are assigned as the 6-bit input for B., with R3 being the most significant bit.

OUTPUTS
The 6 LEDS from (V15-V16) are assigned as the 6-bit output for the ALU, with V15 as the most significant bit.
The LED U14 is assigned as the 1-bit carry out output.
The LED V14 is assigned as the 1-bit overflow output.
The LED V13 is assigned as the 1-bit greater than equal to output.

PREVIOUS SUBMISSIONS

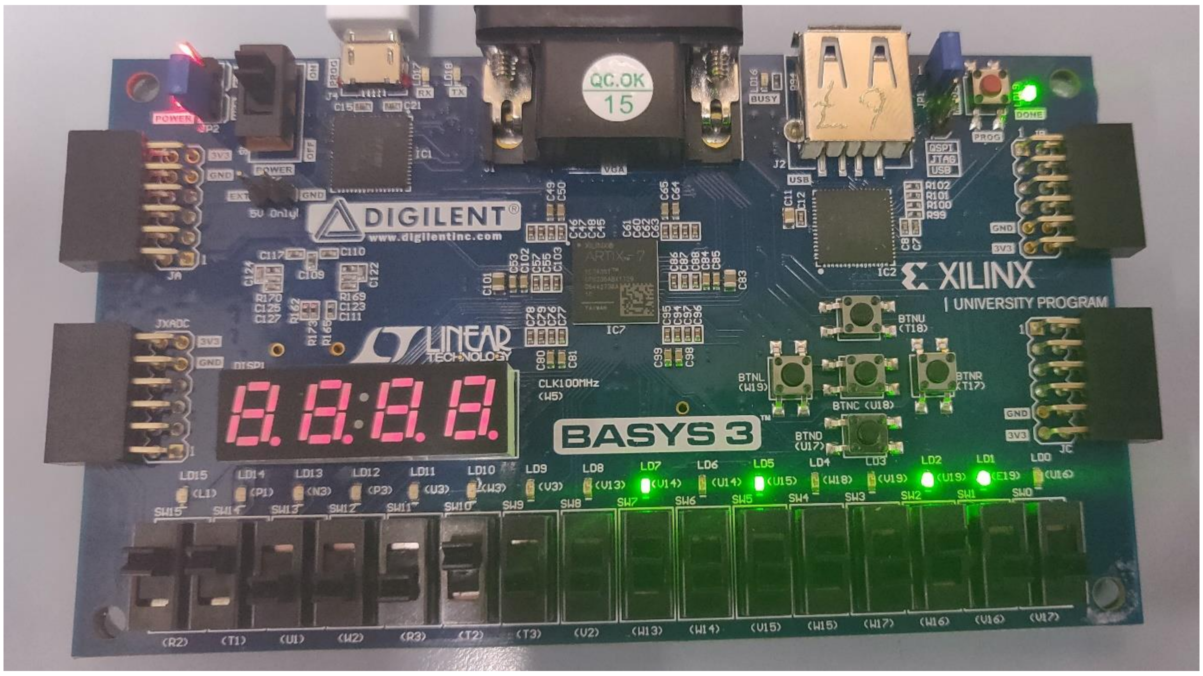
I’ve previously submitted my 6_bit_ripple_adder and my 8_bit_comparator codes on time which I’ve then imported into this project and used as modules.
While I could not import all the testcases I tried previously into this project, I’ve tried most of the testcases here.
I’ve thoroughly tested these modules before submitting them as can be seen from the reports I’ve included herewith.

I’ve included both of my previous files as LabB_kohliv and LabC_kohliv.

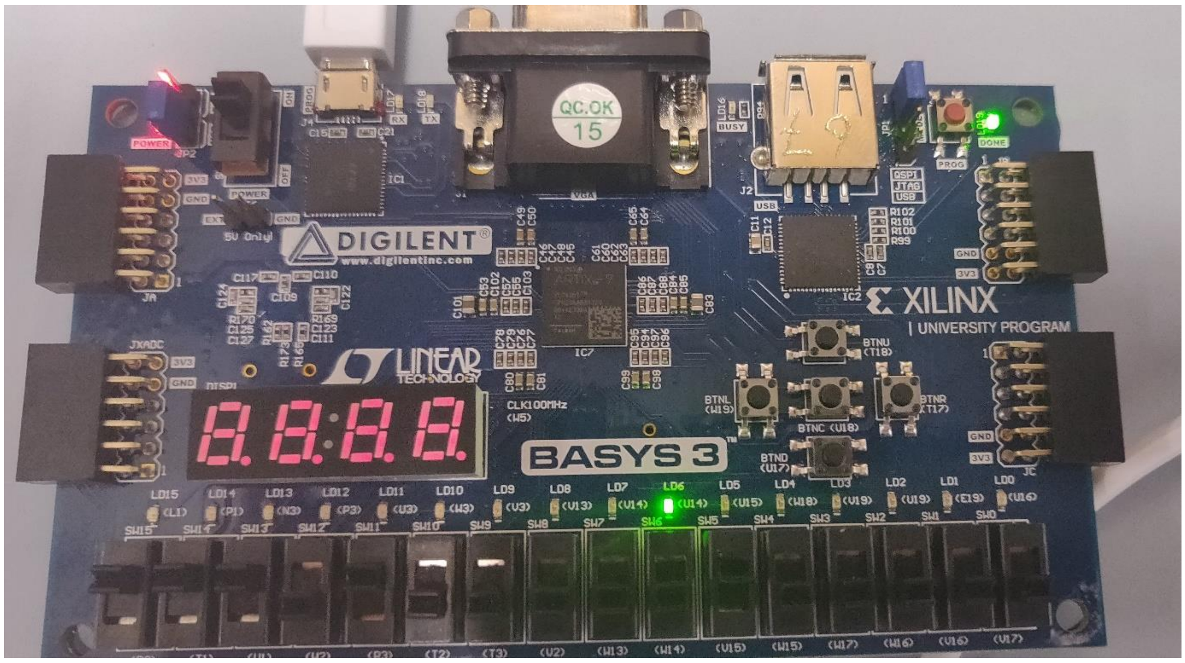
WHAT I WOULD DO DIFFERENTLY

I had a lot of errors due to not initializing outputs before moving to other functions and that is something I had to fix but works perfectly now.
Later, after I’d already completed a large part of my project I thought I could even do the first few function outputs using the adder module but by that I’d have to re-do a large part so I just stuck to defining separate modules for everything.

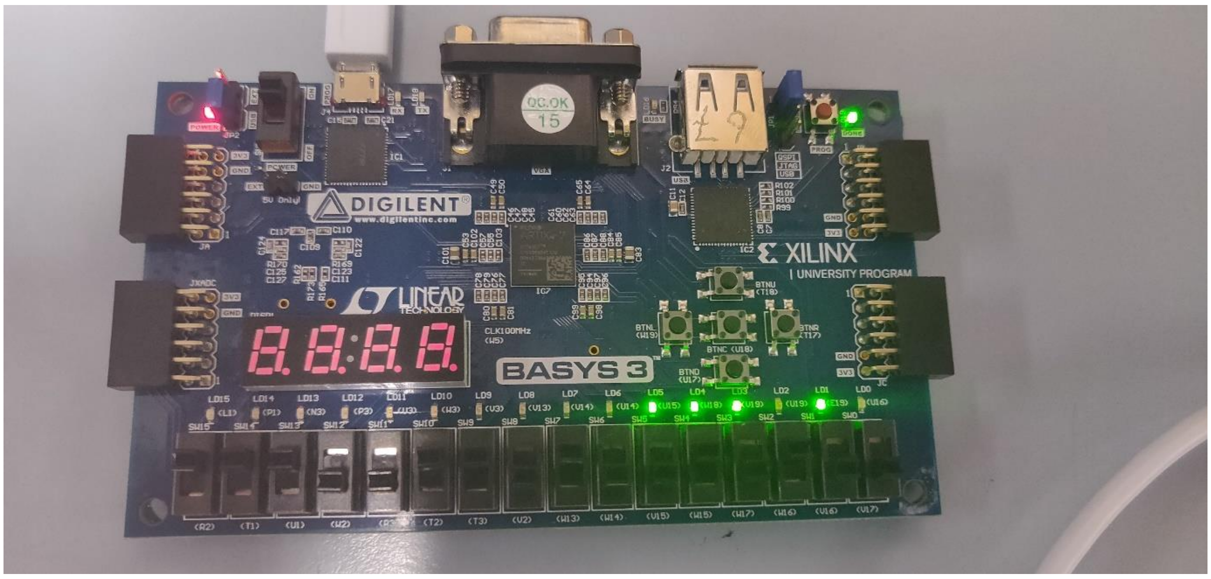
DISPLAYING OUTPUTS

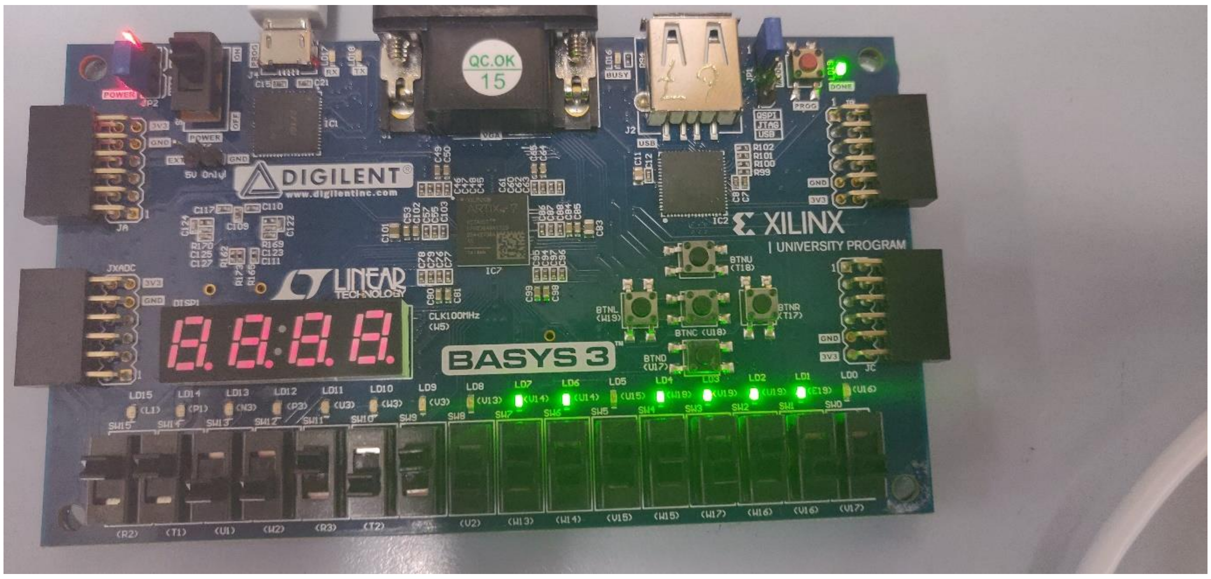


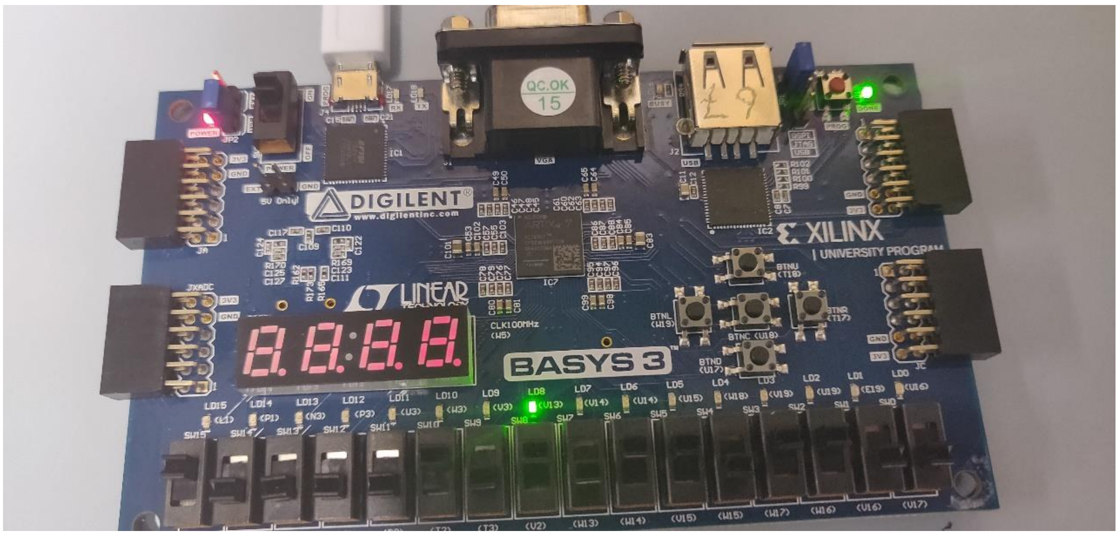
ADDER TESTBENCH CAUSING OVERFLOW

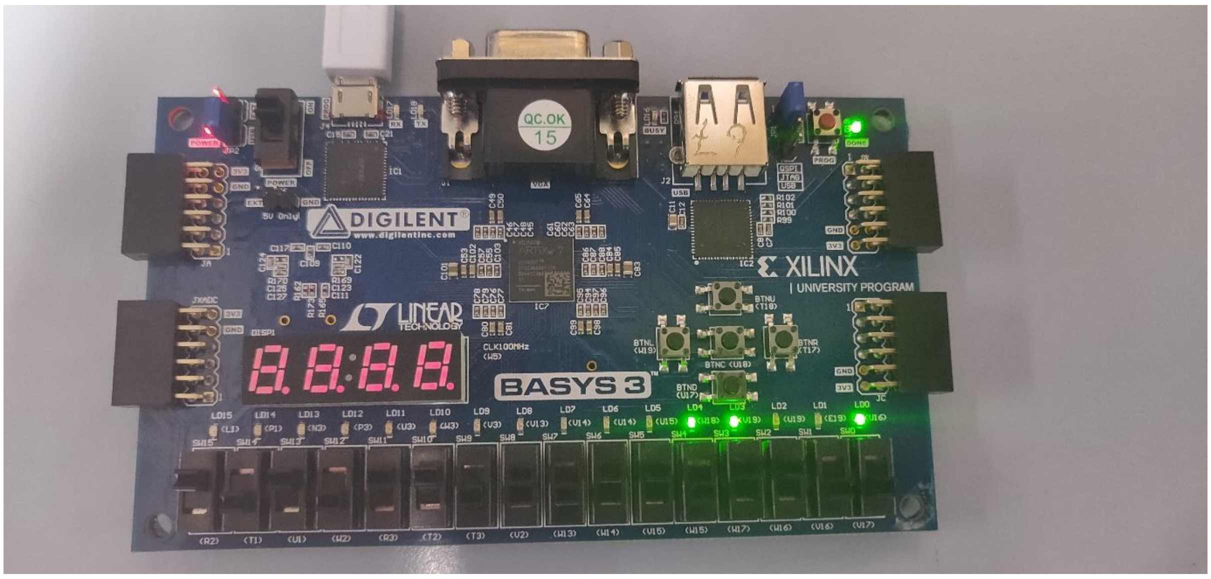


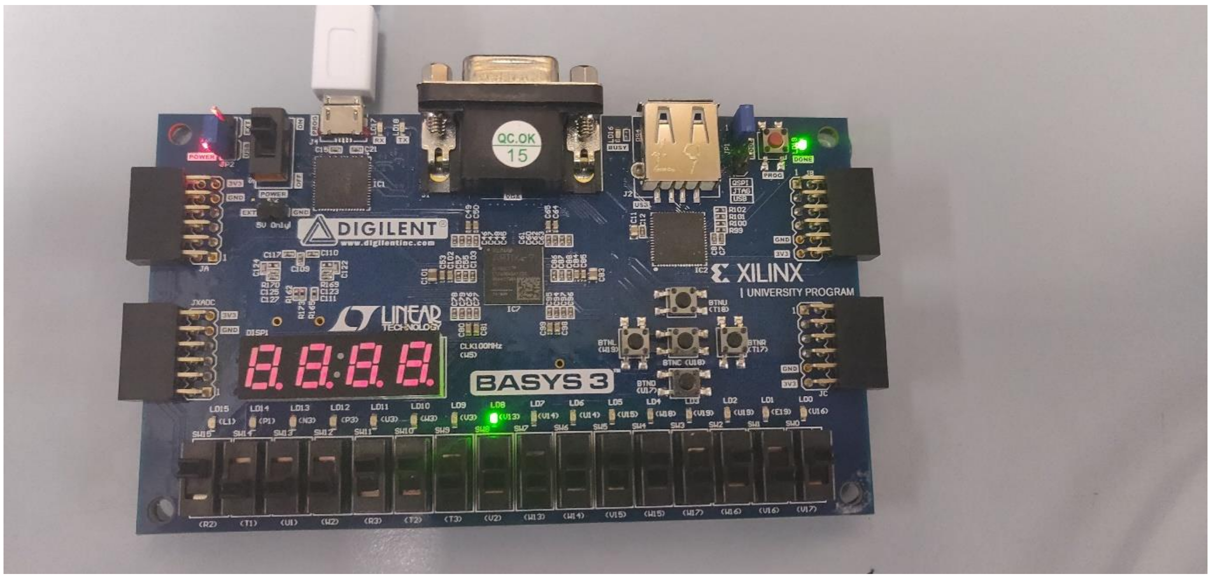
SUBTRACTOR TESTBENCH CAUSING CARRY

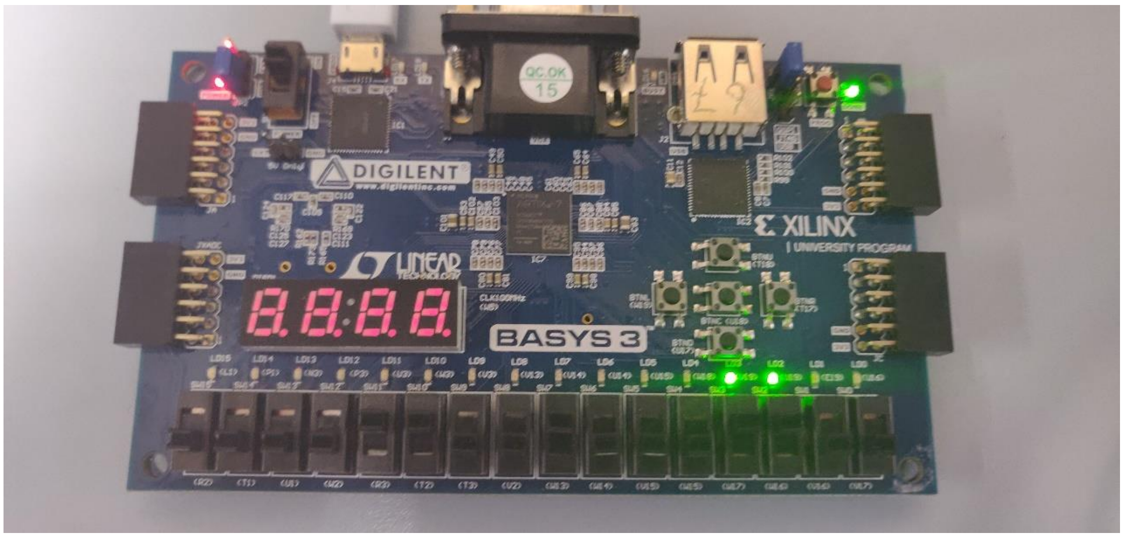

SUBTRACTOR TESTBENCH WITH NO CARRY AND OVERFLOW

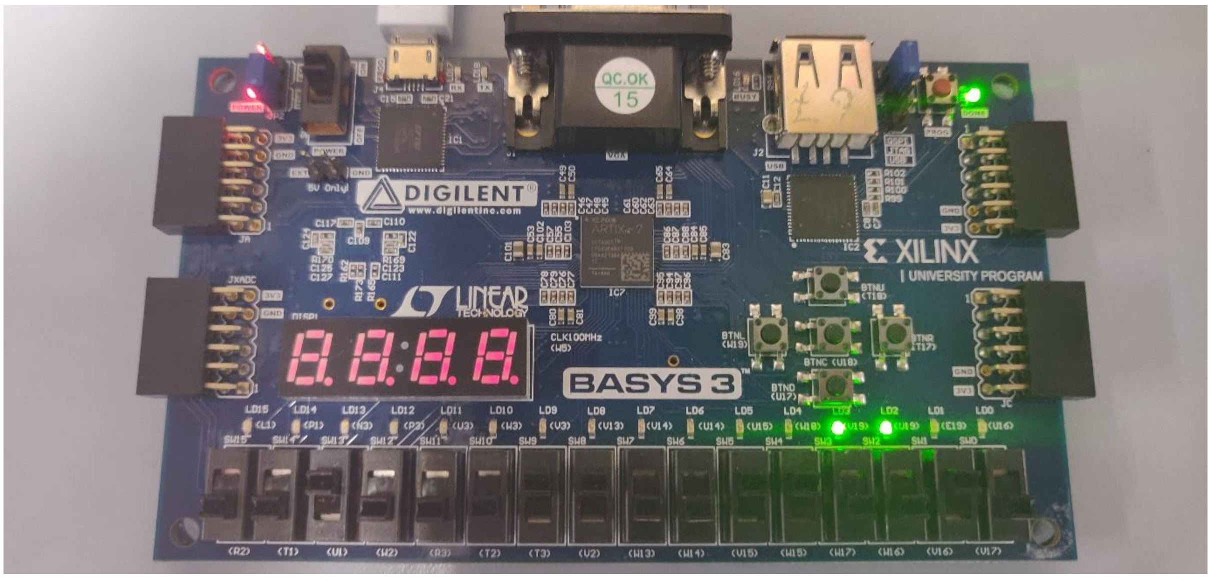

ADDER TESTBENCH WITH CARRY AND OVERFLOW

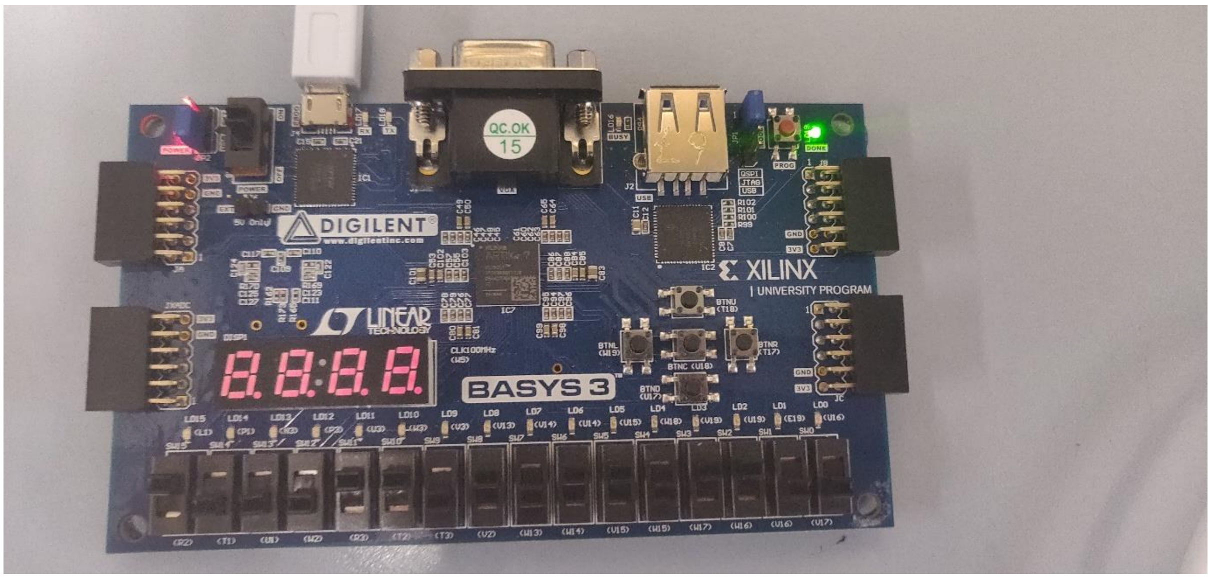

GREATER THAN EQUAL TO WITH ONE POSITIVE AND ONE NEGATIVE

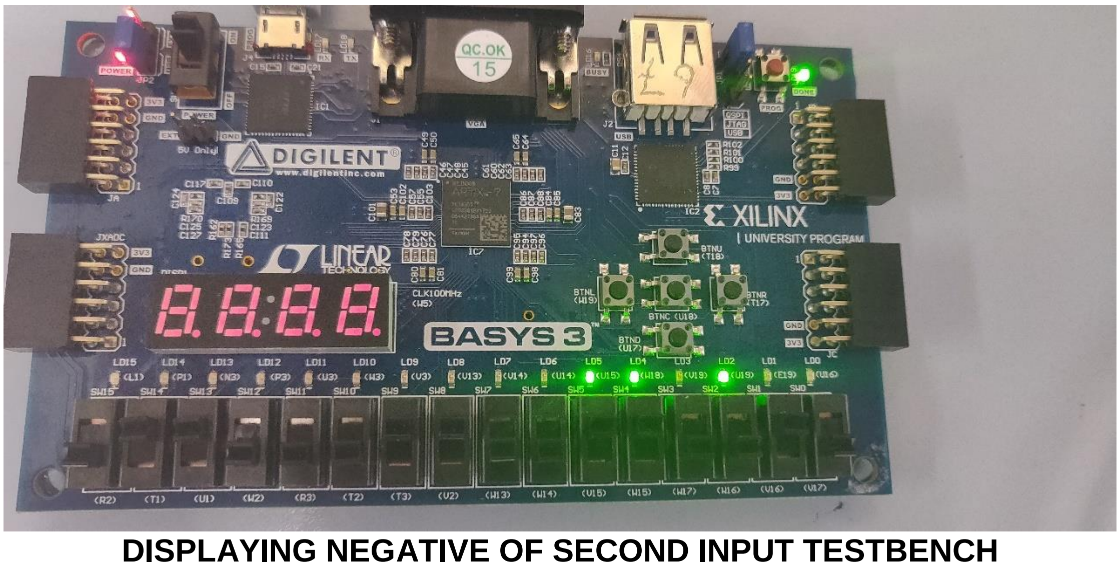

DISPLAYING XOR OF BOTH INPUTS

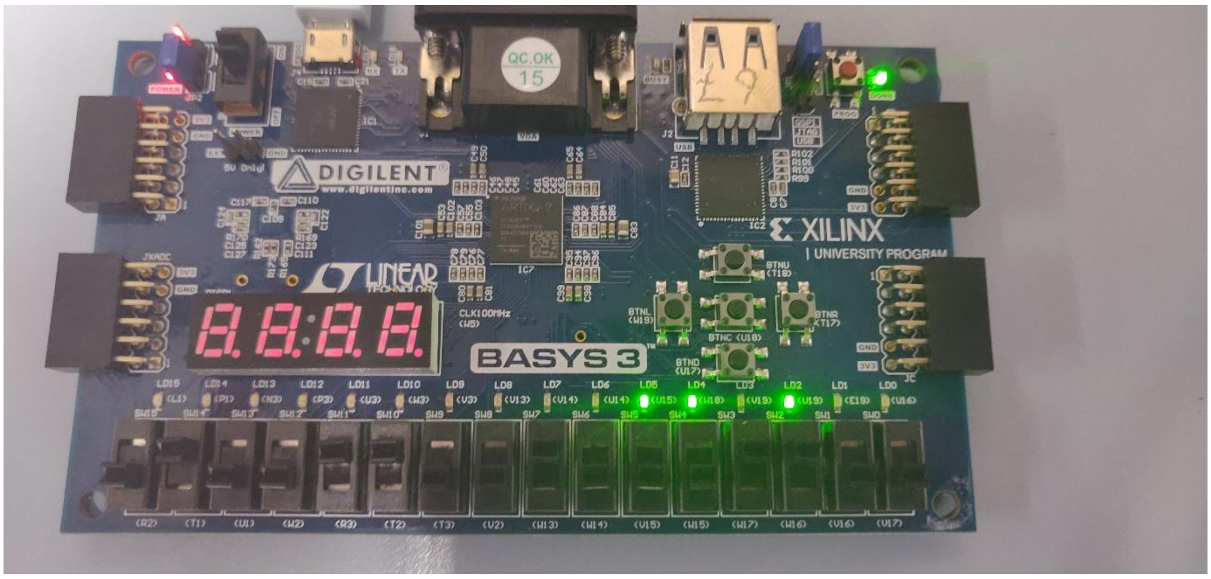

GREATER THAN EQUAL TO WITH BOTH NEGATIVE INPUTS


DISPLAYING FIRST INPUT TESTBENCH


DISPLAYING SECOND INPUT TESTBENCH


GREATER THAN EQUAL TO WITH BOTH POSITIVE INPUTS


DISPLAYING NEGATIVE OF SECOND INPUT TESTBENCH



DISPLAYING NEGATIVE OF FIRST INPUT TESTBENCH