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

3C7 DIGITAL SYSTEMS DESIGN LABORATORY

Assignment 01 Report

Department of Electronic and Electrical Engineering

Professor: Dr. Shreejith Shanker

1. Abstract:

The 3C7 Digital Systems Design assignment involves designing and testing a mini–Arithmetic Logic Unit (ALU) and implementing it on the Basys-3 Board. For an understanding of the development and testing of more complex combinational designs in digital systems, this study synthesizes knowledge from earlier laboratories. The ALU must perform specific functions on two 6-bit inputs in 2's complement format, producing a 6-bit output and various operations based on the input function code. The assignment emphasizes the importance of understanding and applying digital design principles in a practical context to implement a suitable FPGA design.

Submitted By:
Shreshtha Kamboj
23364317
Submission Date: 09/03/2024

2. Introduction:

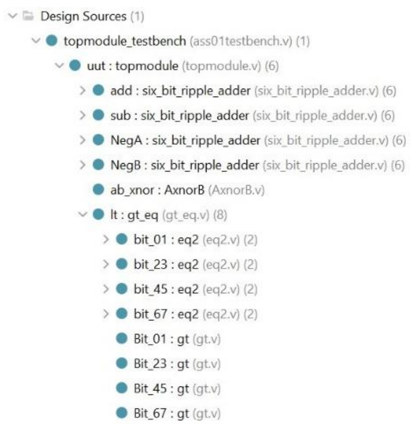
This assignment's objectives were to gather all the knowledge from previous labs to design a Verilog module to implement various functions using an arithmetic logic unit (ALU). An ALU has a varied range of functionality but in this assignment, we use it to take input two 6-bit binary numbers A & B and give output a 6-bit binary number X in 2's complement form. We do both testing it on the Basys 3 Board and running its behavioural simulation to draw certain conclusions by observing its output. We also look at previous lab session's implementation in the appendices section in the end.

3. Implementation:

We implement various logical operations on two 6-bit input binary numbers and use the logic's build in previous lab sessions for implementation in this assignment.

4. Sources:

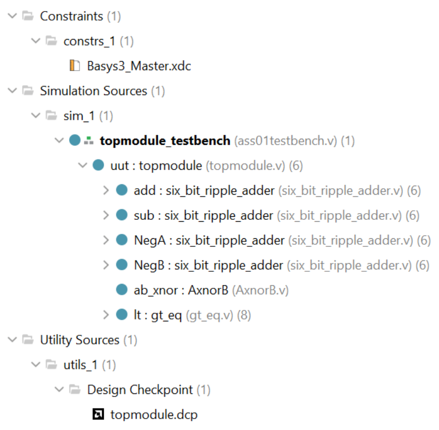
The Design sources consists of topmodule_testbench module which inherits the properties of other instantiations of modules shown below:



```
Design Sources (1)
  topmodule_testbench (ass01testbench.v) (1)
    uut : topmodule (topmodule.v) (6)
      add : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
      sub : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
      NegA : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
      NegB : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
      ab_xnor : AxnorB (AxnorB.v)
      lt : gt_eq (gt_eq.v) (8)
        bit_01 : eq2 (eq2.v) (2)
        bit_23 : eq2 (eq2.v) (2)
        bit_45 : eq2 (eq2.v) (2)
        bit_67 : eq2 (eq2.v) (2)
        Bit_01 : gt (gt.v)
        Bit_23 : gt (gt.v)
        Bit_45 : gt (gt.v)
        Bit_67 : gt (gt.v)
```

Fig 4.1 Design Sources

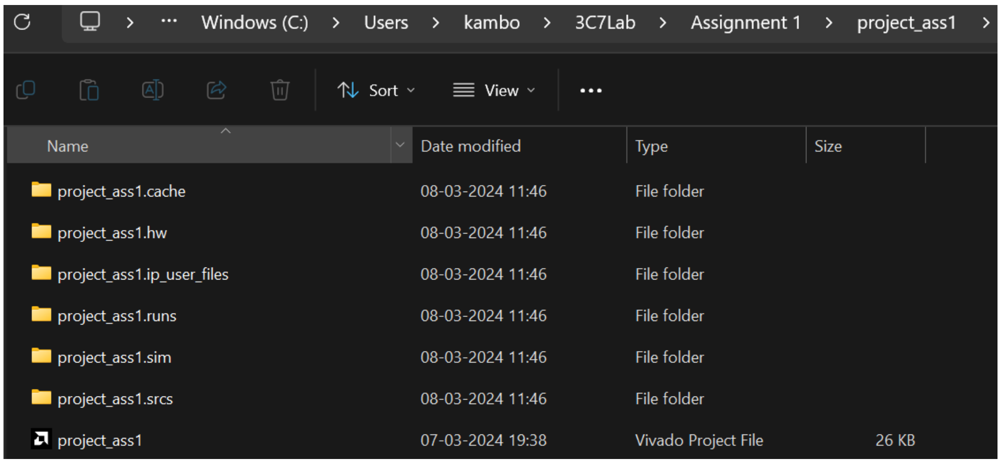
The constraints, Simulation sources, and utility sources are as shown below:



```
Constraints (1)
  constrs_1 (1)
    Basys3_Master.xdc
Simulation Sources (1)
  sim_1 (1)
    topmodule_testbench (ass01testbench.v) (1)
      uut : topmodule (topmodule.v) (6)
        add : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
        sub : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
        NegA : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
        NegB : six_bit_ripple_adder (six_bit_ripple_adder.v) (6)
        ab_xnor : AxnorB (AxnorB.v)
        lt : gt_eq (gt_eq.v) (8)
Utility Sources (1)
  utils_1 (1)
    Design Checkpoint (1)
      topmodule.dcp
```

Fig 4.2 Constraints, Simulation Sources, Utility Sources

File directory is as follows:



Windows (C:) > Users > kambo > 3C7Lab > Assignment 1 > project_ass1 >

Sort View

Name	Date modified	Type	Size
project_ass1.cache	08-03-2024 11:46	File folder	
project_ass1.hw	08-03-2024 11:46	File folder	
project_ass1.ip_user_files	08-03-2024 11:46	File folder	
project_ass1.runs	08-03-2024 11:46	File folder	
project_ass1.sim	08-03-2024 11:46	File folder	
project_ass1.srcs	08-03-2024 11:46	File folder	
project_ass1	07-03-2024 19:38	Vivado Project File	26 KB

Fig 4.3 File Directory

5. Schematics Generated:

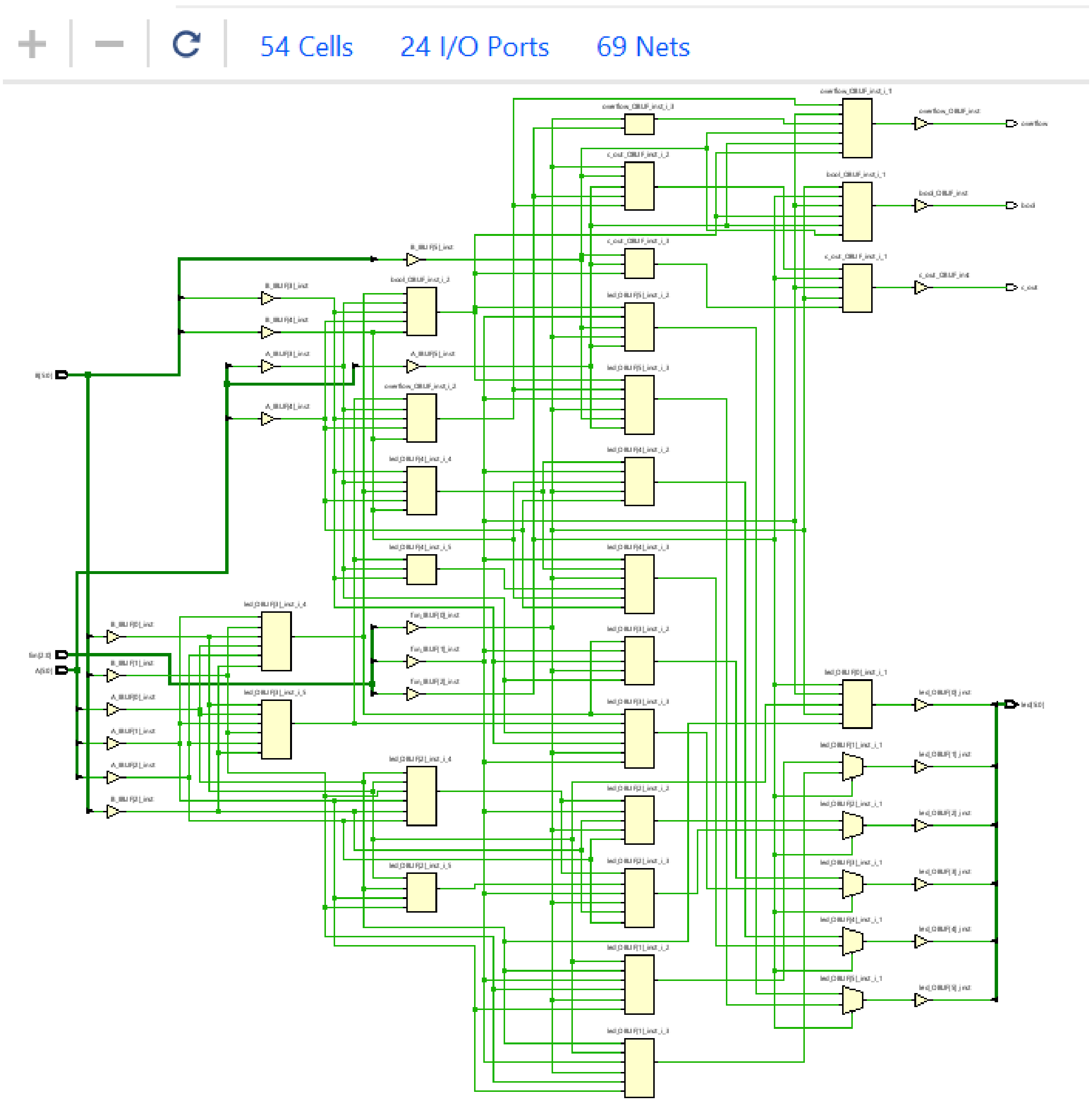


Fig 5.1 Elaborated Design

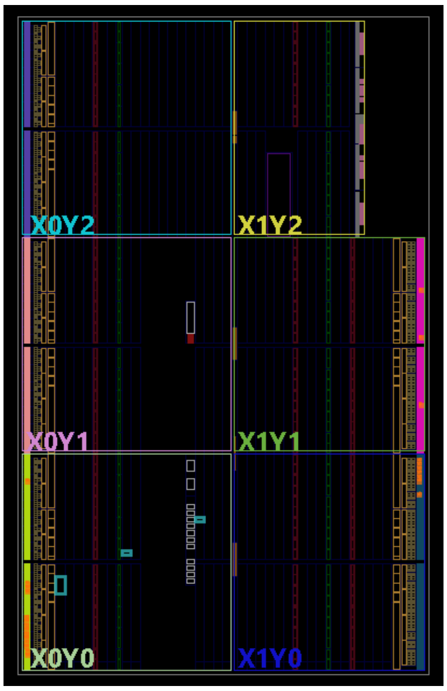


Fig 5.2 Implemented Device

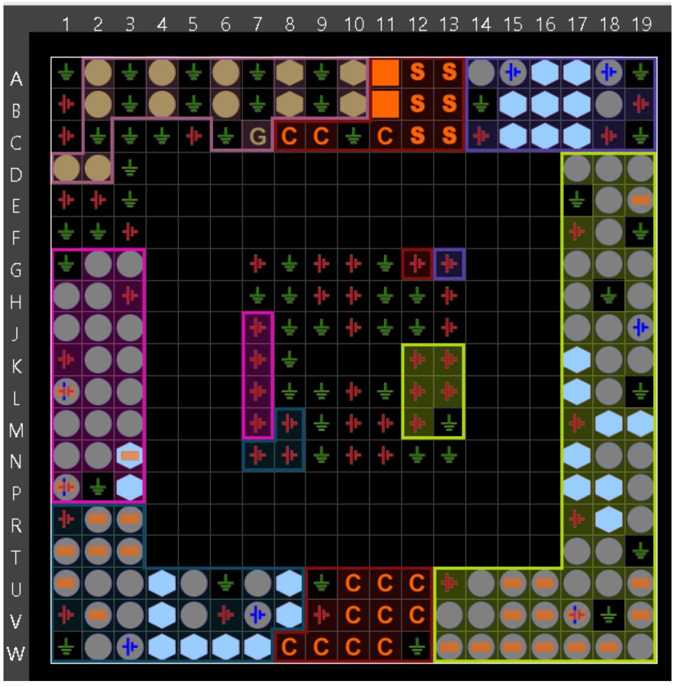
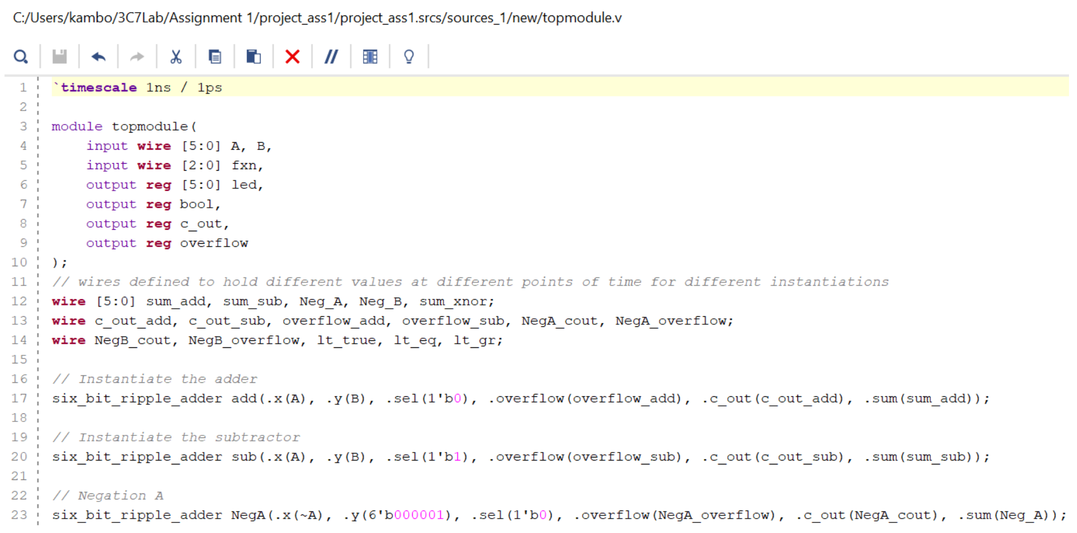


Fig 5.3 Implemented Package

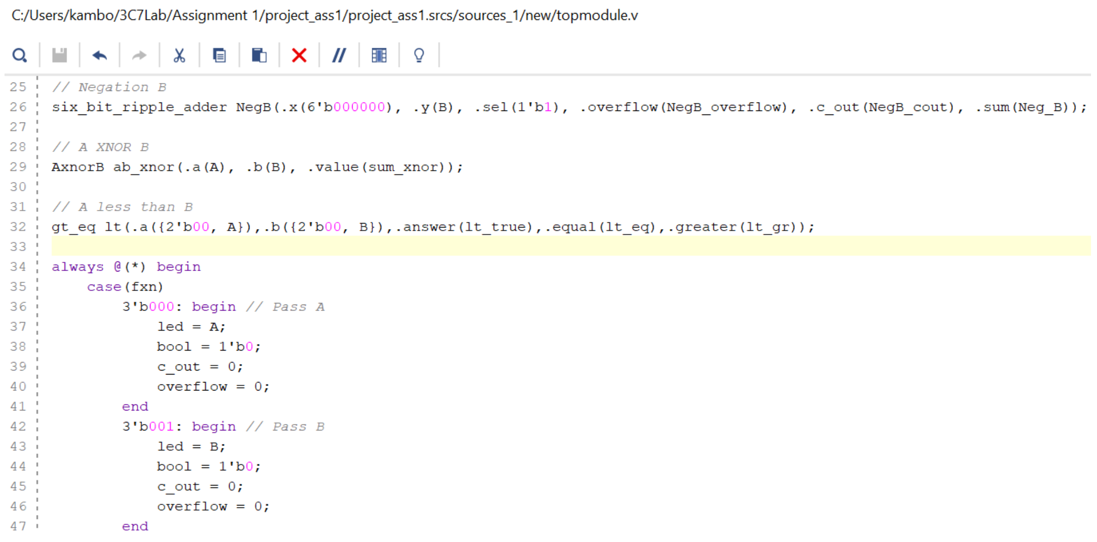
6. Code Snippets:

We have included the following modules in this assignment to correctly implement the given task:-

- topmodules.v
- ass01testbench.v
- six_bit_ripple_adder.v
- AxnorB.v
- gt.v
- gt_eq.v
- eq1.v
- eq2.v
- Basys3_Master.xdc

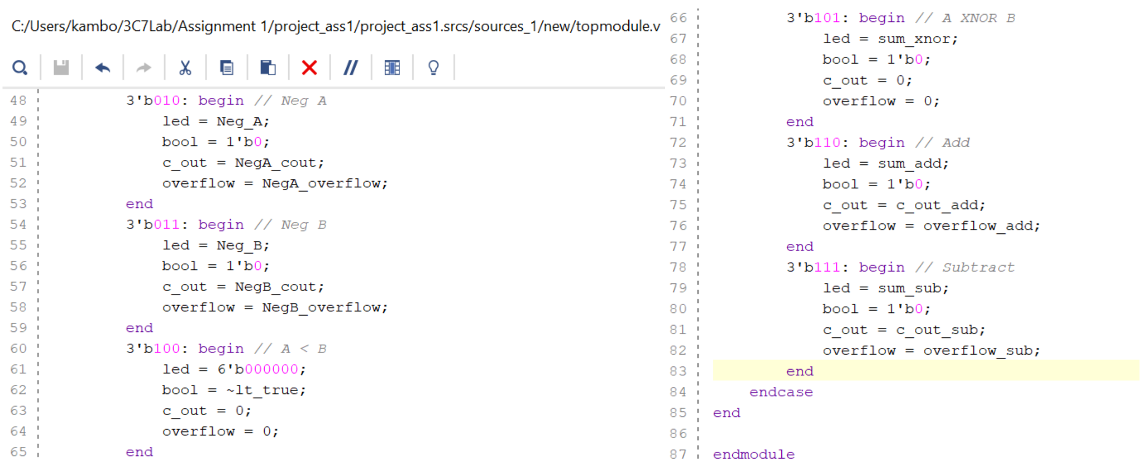

C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/topmodule.v

```verilog
`timescale 1ns / 1ps

module topmodule(
    input wire [5:0] A, B,
    input wire [2:0] fxn,
    output reg [5:0] led,
    output reg bool,
    output reg c_out,
    output reg overflow
);
// wires defined to hold different values at different points of time for different instantiations
wire [5:0] sum_add, sum_sub, Neg_A, Neg_B, sum_xnor;
wire c_out_add, c_out_sub, overflow_add, overflow_sub, NegA_cout, NegA_overflow;
wire NegB_cout, NegB_overflow, lt_true, lt_eq, lt_gr;

// Instantiate the adder
six_bit_ripple_adder add(.x(A), .y(B), .sel(1'b0), .overflow(overflow_add), .c_out(c_out_add), .sum(sum_add));

// Instantiate the subtractor
six_bit_ripple_adder sub(.x(A), .y(B), .sel(1'b1), .overflow(overflow_sub), .c_out(c_out_sub), .sum(sum_sub));

// Negation A
six_bit_ripple_adder NegA(.x(~A), .y(6'b000001), .sel(1'b0), .overflow(NegA_overflow), .c_out(NegA_cout), .sum(Neg_A));
```



C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/topmodule.v

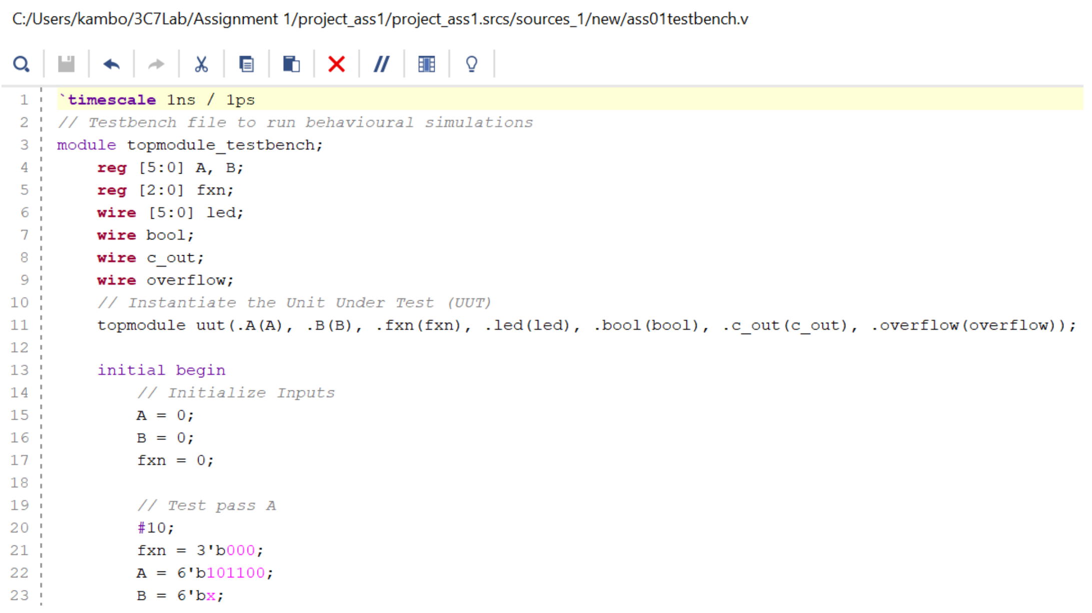
```
// Negation B
six_bit_ripple_adder NegB(.x(6'b000000), .y(B), .sel(1'b1), .overflow(NegB_overflow), .c_out(NegB_cout), .sum(Neg_B));

// A XNOR B
AxnorB ab_xnor(.a(A), .b(B), .value(sum_xnor));

// A less than B
gt_eq lt(.a({2'b00, A}),.b({2'b00, B}),.answer(lt_true),.equal(lt_eq),.greater(lt_gr));

always @(*) begin
    case(fxn)
        3'b000: begin // Pass A
            led = A;
            bool = 1'b0;
            c_out = 0;
            overflow = 0;
        end
        3'b001: begin // Pass B
            led = B;
            bool = 1'b0;
            c_out = 0;
            overflow = 0;
        end
```



C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/topmodule.v

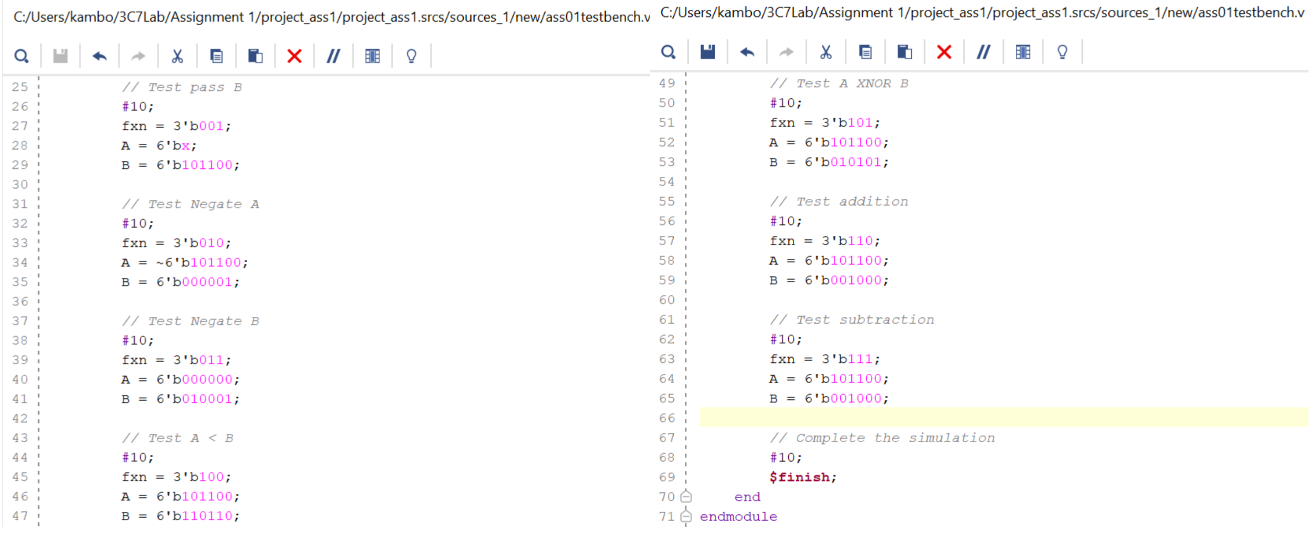
```
        3'b010: begin // Neg A
            led = Neg_A;
            bool = 1'b0;
            c_out = NegA_cout;
            overflow = NegA_overflow;
        end
        3'b011: begin // Neg B
            led = Neg_B;
            bool = 1'b0;
            c_out = NegB_cout;
            overflow = NegB_overflow;
        end
        3'b100: begin // A < B
            led = 6'b000000;
            bool = ~lt_true;
            c_out = 0;
            overflow = 0;
        end
        3'b101: begin // A XNOR B
            led = sum_xnor;
            bool = 1'b0;
            c_out = 0;
            overflow = 0;
        end
        3'b110: begin // Add
            led = sum_add;
            bool = 1'b0;
            c_out = c_out_add;
            overflow = overflow_add;
        end
        3'b111: begin // Subtract
            led = sum_sub;
            bool = 1'b0;
            c_out = c_out_sub;
            overflow = overflow_sub;
        end
    endcase
end

endmodule
```

Fig 6.1 Module having instantiations of other modules



C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/ass01testbench.v

```
`timescale 1ns / 1ps
// Testbench file to run behavioural simulations
module topmodule_testbench;
    reg [5:0] A, B;
    reg [2:0] fxn;
    wire [5:0] led;
    wire bool;
    wire c_out;
    wire overflow;
    // Instantiate the Unit Under Test (UUT)
    topmodule uut(.A(A), .B(B), .fxn(fxn), .led(led), .bool(bool), .c_out(c_out), .overflow(overflow));

    initial begin
        // Initialize Inputs
        A = 0;
        B = 0;
        fxn = 0;

        // Test pass A
        #10;
        fxn = 3'b000;
        A = 6'b101100;
        B = 6'bx;
```



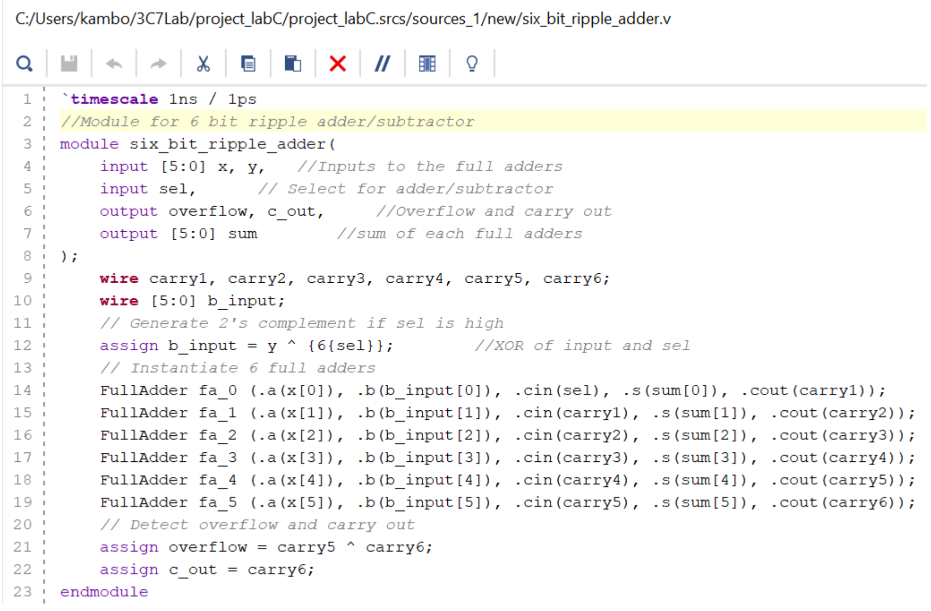
C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/ass01testbench.v

```
        // Test pass B
        #10;
        fxn = 3'b001;
        A = 6'bx;
        B = 6'b101100;

        // Test Negate A
        #10;
        fxn = 3'b010;
        A = ~6'b101100;
        B = 6'b000001;

        // Test Negate B
        #10;
        fxn = 3'b011;
        A = 6'b000000;
        B = 6'b010001;

        // Test A < B
        #10;
        fxn = 3'b100;
        A = 6'b101100;
        B = 6'b110110;
```

C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/ass01testbench.v

```
        // Test A XNOR B
        #10;
        fxn = 3'b101;
        A = 6'b101100;
        B = 6'b010101;

        // Test addition
        #10;
        fxn = 3'b110;
        A = 6'b101100;
        B = 6'b001000;

        // Test subtraction
        #10;
        fxn = 3'b111;
        A = 6'b101100;
        B = 6'b001000;

        // Complete the simulation
        #10;
        $finish;
    end
endmodule
```

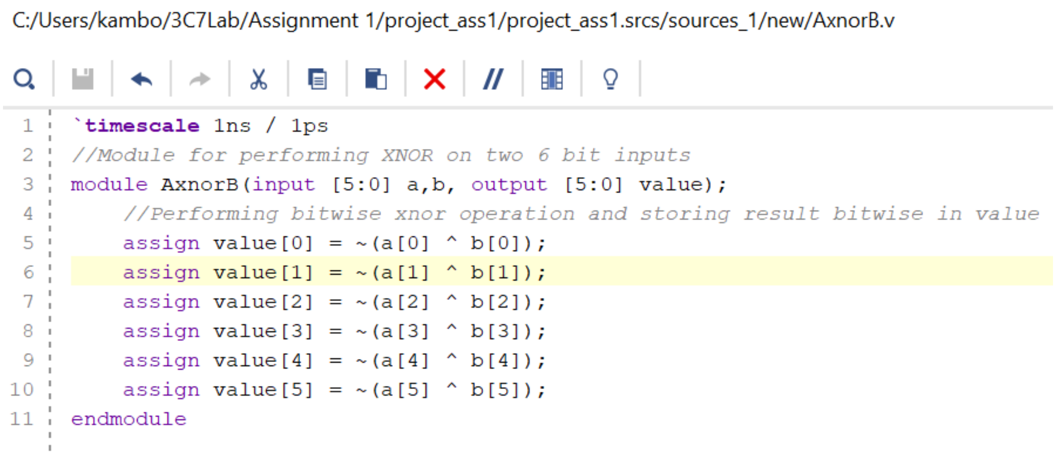
Fig 6.2 Testbench Module having test vectors



C:/Users/kambo/3C7Lab/project_labC/project_labC.srcs/sources_1/new/six_bit_ripple_adder.v

```
`timescale 1ns / 1ps
//Module for 6 bit ripple adder/subtractor
module six_bit_ripple_adder(
    input [5:0] x, y,   //Inputs to the full adders
    input sel,      // Select for adder/subtractor
    output overflow, c_out,    //Overflow and carry out
    output [5:0] sum        //sum of each full adders
);
    wire carry1, carry2, carry3, carry4, carry5, carry6;
    wire [5:0] b_input;
    // Generate 2's complement if sel is high
    assign b_input = y ^ {6{sel}};        //XOR of input and sel
    // Instantiate 6 full adders
    FullAdder fa_0 (.a(x[0]), .b(b_input[0]), .cin(sel), .s(sum[0]), .cout(carry1));
    FullAdder fa_1 (.a(x[1]), .b(b_input[1]), .cin(carry1), .s(sum[1]), .cout(carry2));
    FullAdder fa_2 (.a(x[2]), .b(b_input[2]), .cin(carry2), .s(sum[2]), .cout(carry3));
    FullAdder fa_3 (.a(x[3]), .b(b_input[3]), .cin(carry3), .s(sum[3]), .cout(carry4));
    FullAdder fa_4 (.a(x[4]), .b(b_input[4]), .cin(carry4), .s(sum[4]), .cout(carry5));
    FullAdder fa_5 (.a(x[5]), .b(b_input[5]), .cin(carry5), .s(sum[5]), .cout(carry6));
    // Detect overflow and carry out
    assign overflow = carry5 ^ carry6;
    assign c_out = carry6;
endmodule
```

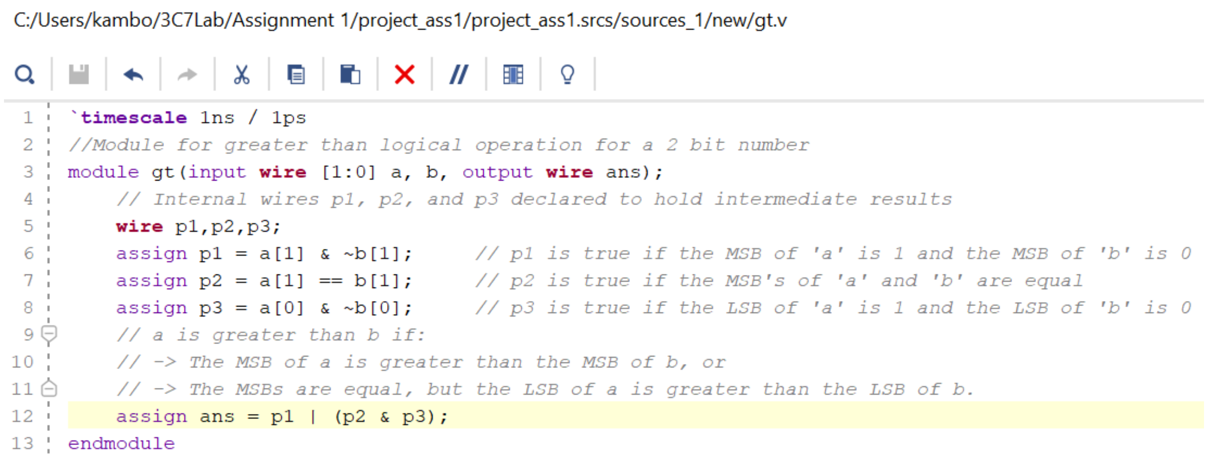
Fig 6.3 Module having 6 instantiations of 1-bit Full Adder



C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/AxnorB.v

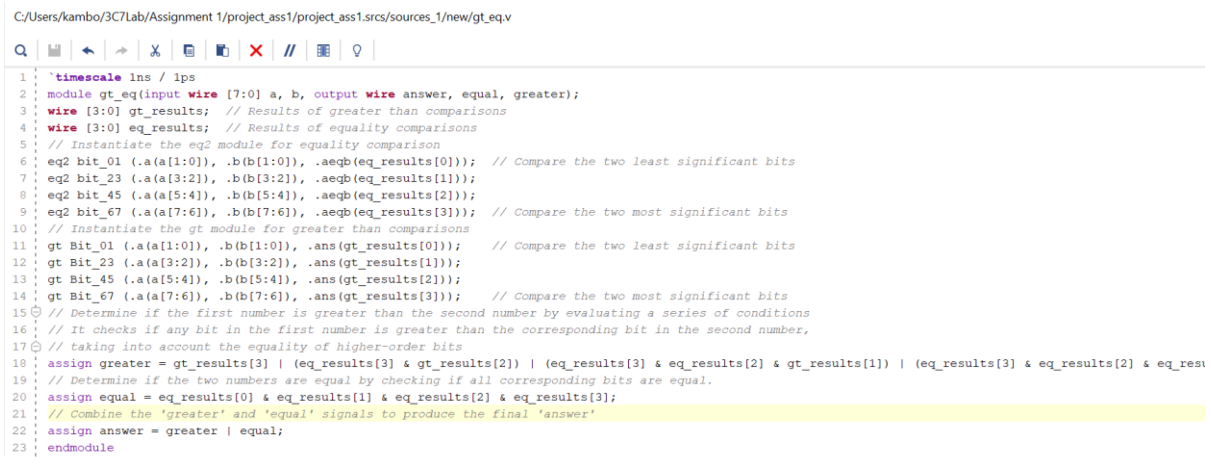
```
`timescale 1ns / 1ps
//Module for performing XNOR on two 6 bit inputs
module AxnorB(input [5:0] a,b, output [5:0] value);
    //Performing bitwise xnor operation and storing result bitwise in value
    assign value[0] = ~(a[0] ^ b[0]);
    assign value[1] = ~(a[1] ^ b[1]);
    assign value[2] = ~(a[2] ^ b[2]);
    assign value[3] = ~(a[3] ^ b[3]);
    assign value[4] = ~(a[4] ^ b[4]);
    assign value[5] = ~(a[5] ^ b[5]);
endmodule
```

Fig 6.4 Module having code for xnor operation


C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/gt.v

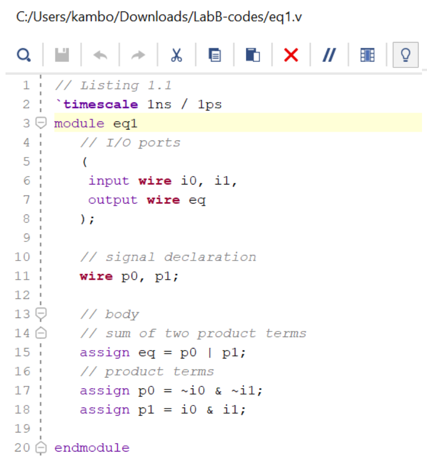
```verilog
`timescale 1ns / 1ps
//Module for greater than logical operation for a 2 bit number
module gt(input wire [1:0] a, b, output wire ans);
    // Internal wires p1, p2, and p3 declared to hold intermediate results
    wire p1,p2,p3;
    assign p1 = a[1] & ~b[1];     // p1 is true if the MSB of 'a' is 1 and the MSB of 'b' is 0
    assign p2 = a[1] == b[1];     // p2 is true if the MSB's of 'a' and 'b' are equal
    assign p3 = a[0] & ~b[0];     // p3 is true if the LSB of 'a' is 1 and the LSB of 'b' is 0
    // a is greater than b if:
    // -> The MSB of a is greater than the MSB of b, or
    // -> The MSBs are equal, but the LSB of a is greater than the LSB of b.
    assign ans = p1 | (p2 & p3);
endmodule
```

Fig 6.5 Module having code for 2-bit greater than circuit

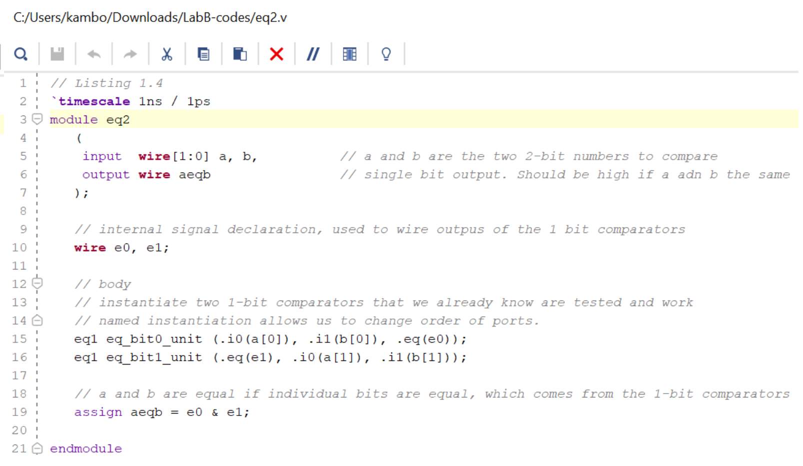

C:/Users/kambo/3C7Lab/Assignment 1/project_ass1/project_ass1.srcs/sources_1/new/gt_eq.v

```verilog
`timescale 1ns / 1ps
module gt_eq(input wire [7:0] a, b, output wire answer, equal, greater);
wire [3:0] gt_results;  // Results of greater than comparisons
wire [3:0] eq_results;  // Results of equality comparisons
// Instantiate the eq2 module for equality comparison
eq2 bit_01 (.a(a[1:0]), .b(b[1:0]), .aeqb(eq_results[0]));  // Compare the two least significant bits
eq2 bit_23 (.a(a[3:2]), .b(b[3:2]), .aeqb(eq_results[1]));
eq2 bit_45 (.a(a[5:4]), .b(b[5:4]), .aeqb(eq_results[2]));
eq2 bit_67 (.a(a[7:6]), .b(b[7:6]), .aeqb(eq_results[3]));  // Compare the two most significant bits
// Instantiate the gt module for greater than comparisons
gt Bit_01 (.a(a[1:0]), .b(b[1:0]), .ans(gt_results[0]));    // Compare the two least significant bits
gt Bit_23 (.a(a[3:2]), .b(b[3:2]), .ans(gt_results[1]));
gt Bit_45 (.a(a[5:4]), .b(b[5:4]), .ans(gt_results[2]));
gt Bit_67 (.a(a[7:6]), .b(b[7:6]), .ans(gt_results[3]));    // Compare the two most significant bits
// Determine if the first number is greater than the second number by evaluating a series of conditions
// It checks if any bit in the first number is greater than the corresponding bit in the second number,
// taking into account the equality of higher-order bits
assign greater = gt_results[3] | (eq_results[3] & gt_results[2]) | (eq_results[3] & eq_results[2] & gt_results[1]) | (eq_results[3] & eq_results[2] & eq_resu
// Determine if the two numbers are equal by checking if all corresponding bits are equal.
assign equal = eq_results[0] & eq_results[1] & eq_results[2] & eq_results[3];
// Combine the 'greater' and 'equal' signals to produce the final 'answer'
assign answer = greater | equal;
endmodule
```

Fig 6.6 Module having code for 8-bit greater than circuit


C:/Users/kambo/Downloads/LabB-codes/eq1.v

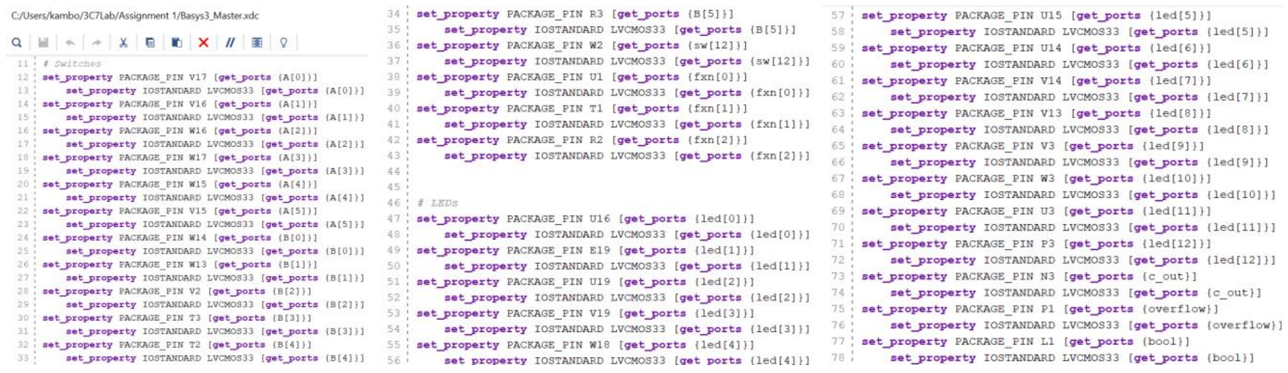
```verilog
// Listing 1.1
`timescale 1ns / 1ps
module eq1
   // I/O ports
   (
    input wire i0, i1,
    output wire eq
   );

   // 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
```


C:/Users/kambo/Downloads/LabB-codes/eq2.v

```verilog
// Listing 1.4
`timescale 1ns / 1ps
module eq2
   (
    input  wire[1:0] a, b,          // a and 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
```

Fig 6.7 Module for 1-bit comparator and 2-bit comparator

The following image shows the constraints for Basys 3 board implementation. The implementation will be explained in the demonstration section.



```
# Switches
set_property PACKAGE_PIN V17 [get_ports {A[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
set_property PACKAGE_PIN V16 [get_ports {A[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]
set_property PACKAGE_PIN W16 [get_ports {A[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]
set_property PACKAGE_PIN W17 [get_ports {A[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]
set_property PACKAGE_PIN W15 [get_ports {A[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A[4]}]
set_property PACKAGE_PIN V15 [get_ports {A[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A[5]}]
set_property PACKAGE_PIN W14 [get_ports {B[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]
set_property PACKAGE_PIN W13 [get_ports {B[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]
set_property PACKAGE_PIN V2 [get_ports {B[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]
set_property PACKAGE_PIN T3 [get_ports {B[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]
set_property PACKAGE_PIN T2 [get_ports {B[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {B[4]}]
set_property PACKAGE_PIN R3 [get_ports {B[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {B[5]}]
set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
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]}]


# LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
set_property PACKAGE_PIN W3 [get_ports {led[10]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
set_property PACKAGE_PIN U3 [get_ports {led[11]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
set_property PACKAGE_PIN P3 [get_ports {led[12]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
set_property PACKAGE_PIN N3 [get_ports {c_out}]
    set_property IOSTANDARD LVCMOS33 [get_ports {c_out}]
set_property PACKAGE_PIN P1 [get_ports {overflow}]
    set_property IOSTANDARD LVCMOS33 [get_ports {overflow}]
set_property PACKAGE_PIN L1 [get_ports {bool}]
    set_property IOSTANDARD LVCMOS33 [get_ports {bool}]
```

Fig 6.8 Constraints File for Basys 3 Board

7. Demonstration

Configuration Device: BASYS 3 (xc7a35tcpg236-1)
Target Language: Verilog

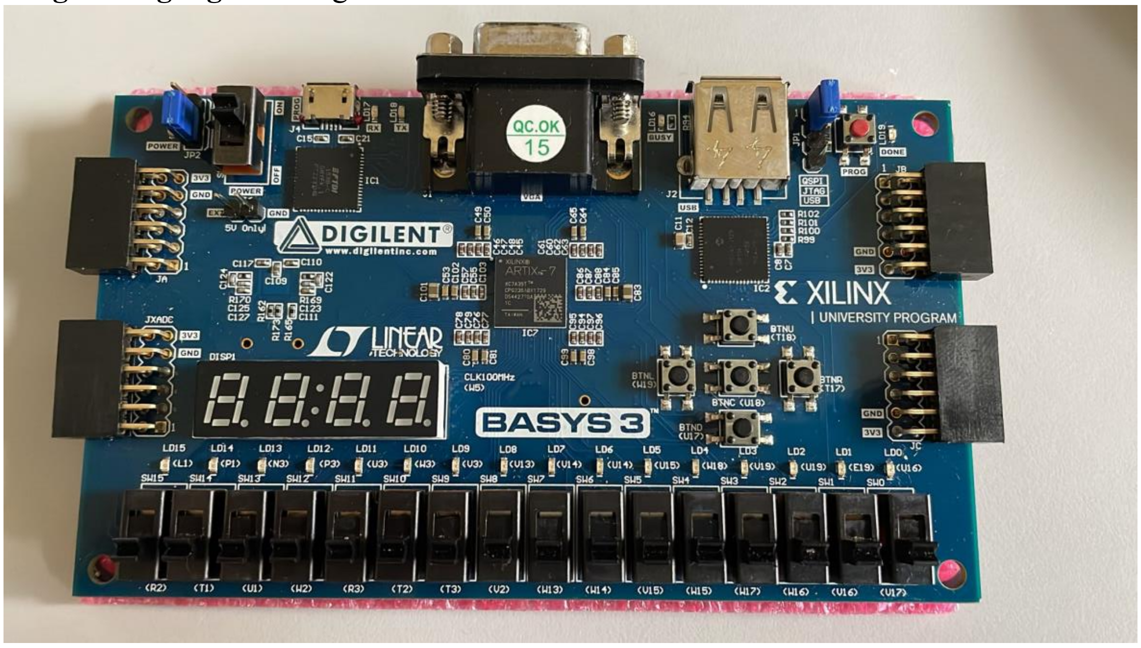


Fig 7.1 Basys 3 Board

SWITCHES - INPUT				LED - INPUT	
V17	A[0]	V2	B[2]	U16	LED[0]
V16	A[1]	T3	B[3]	E19	LED [1]
W16	A[2]	T2	B[4]	U19	LED [2]
W17	A[3]	R3	B[5]	V19	LED [3]
W15	A[4]	U1	fxn[0]	W18	LED [4]
V15	A[5]	T1	fxn[1]	U15	LED [5]
W14	B[0]	R2	fxn[2]	L1	c_out
W13	B[1]	N3	bool	P1	overflow

Table 7.1 Input switched & Output LEDs

The input is taken using the switches assigned. State ‘0’ means OFF and state ‘1’ means ON. The output is generated using LEDs assigned. State ‘0’ means OFF and state ‘1’ means ON.

For fxn values 3’b000, 3’b001, 3’b010, 3’b011, 3’b110 and 3’b111 we have reused the previous code which was implemented in LAB C by instantiating it for different arithmetic and logical operations to be performed in this assignment.

For fxn value 3’b100 we have created a separate module to implement the required logic.

For fxn value 3’b101 we have reused the previous code which was implemented in LAB B by instantiating it for the logic required to be implemented here.

First, we define all the logics across all the modules and then we create a topmodule to work with all these other modules by instantiating them. After all this we generate the bit stream which automatically synthesizes and implements the design. Then we connect the board using open target in hardware manager and after it is connected, we just program the device.

Board Number = 44
Binary Equivalent = 6b’101100

Test Case 1: Displaying A
fxn = 3’b000
Test Vector A = 6'b101100
Test Vector B = 6’bx
Sel = 1’b0

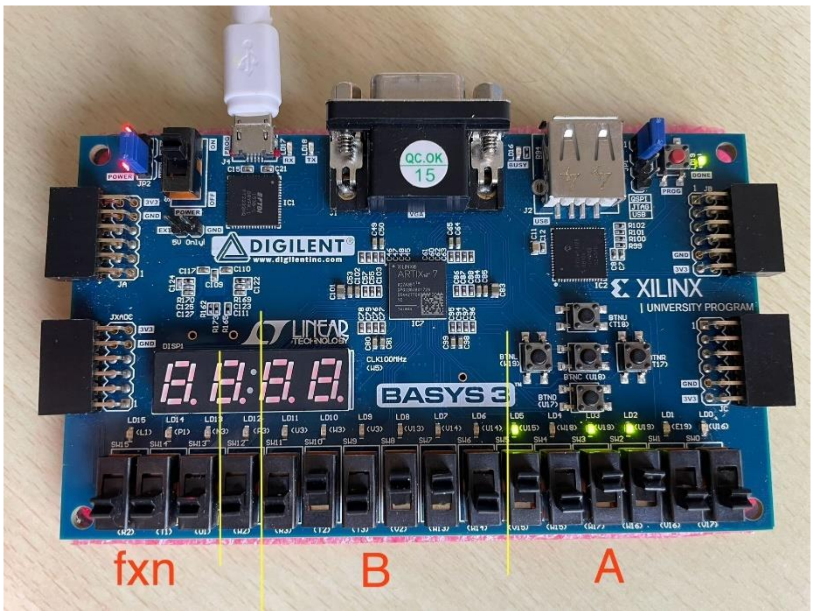


Fig 7.2 fxn = 3’b000 output on Basys 3 Board

Here we observe that whatever value we give to A, it is shown in the output on board irrespective of the value of B. Like we have given A the decimal value 44 in binary and B some random value but we observe that value of A is output on the board by the first six LEDs.

Test Case 2: Displaying B

fxn = 3’b001

Test Vector A = 6’bx

Test Vector B = 6'b101100

Sel = 1’b0

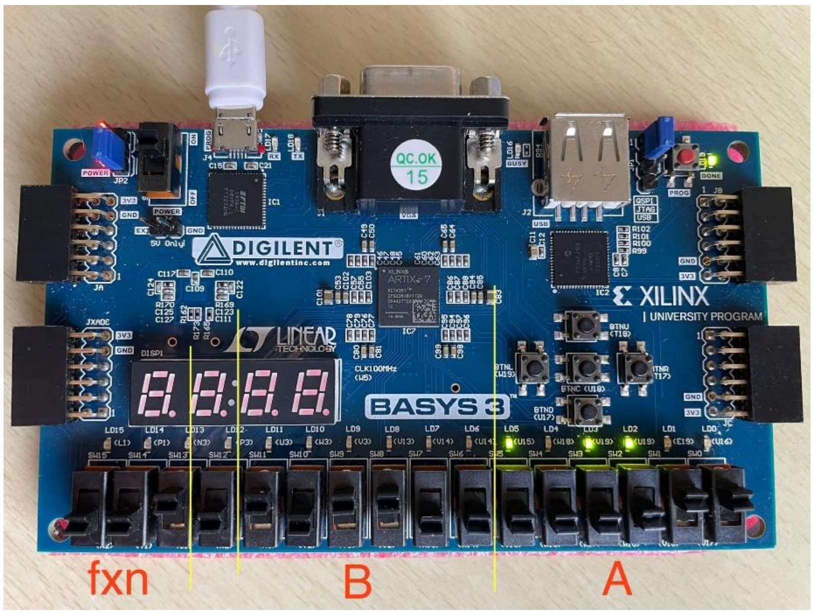


Fig 7.3 fxn = 3’b001 output on Basys 3 Board

Here we observe that whatever value we give to B, it is shown in the output on board irrespective of the value of A. Like we have given B the decimal value 44 in binary and A some random value but we observe that value of A is output on the board by the first six LEDs.

Test Case 3: Displaying Negation of A

fxn = 3’b010

Test Vector A = ~6'b101100

Test Vector B = 6'b000001

Sel = 1’b0

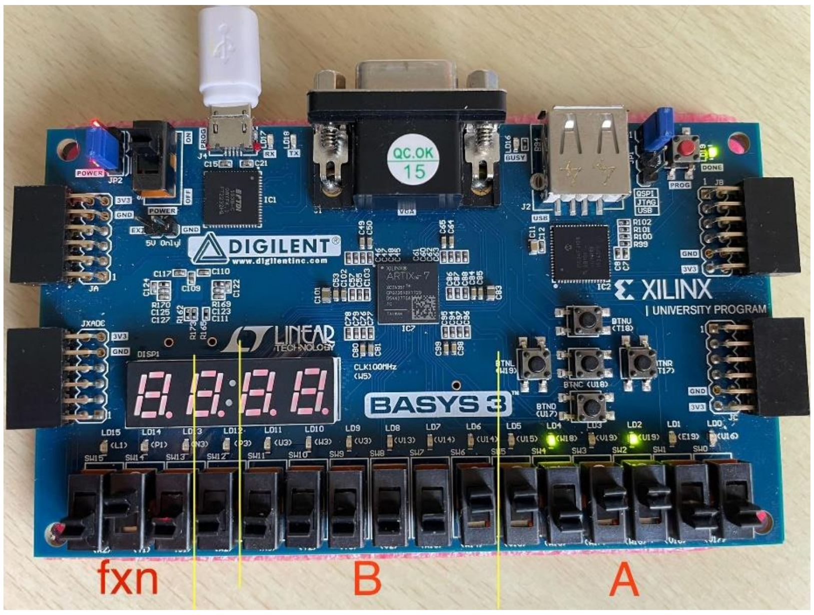


Fig 7.4 fxn = 3’b010 output on Basys 3 Board

Here we observe that whatever value we give to A, the negation of that value is shown on the board. Like we have given A the decimal value 44 in binary and B the decimal value 1 in binary but we observe that 2's complement value of A is output on the board by the first six LEDs.

Test Case 4: Displaying Negation of B

fxn = 3'b011
Test Vector A = 6'b000000
Test Vector B = 6'b010001
Sel = 1'b1

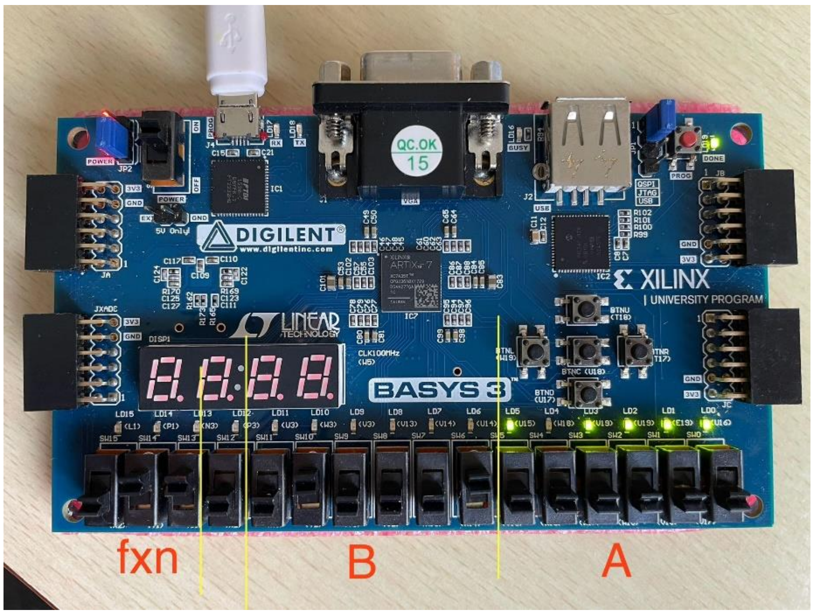


Fig 7.5 fxn = 3'b011 output on Basys 3 Board

Here we observe that whatever value we give to B, the negation of that value is shown on the board. Like we have given B the decimal value 17 in binary and A the decimal value 0 in binary but we observe that 2's complement value of B is output on the board by the first six LEDs.

Test Case 5: Displaying A less than B

fxn = 3'b100
Test Vector A = 6'b101100
Test Vector B = 6'b110110

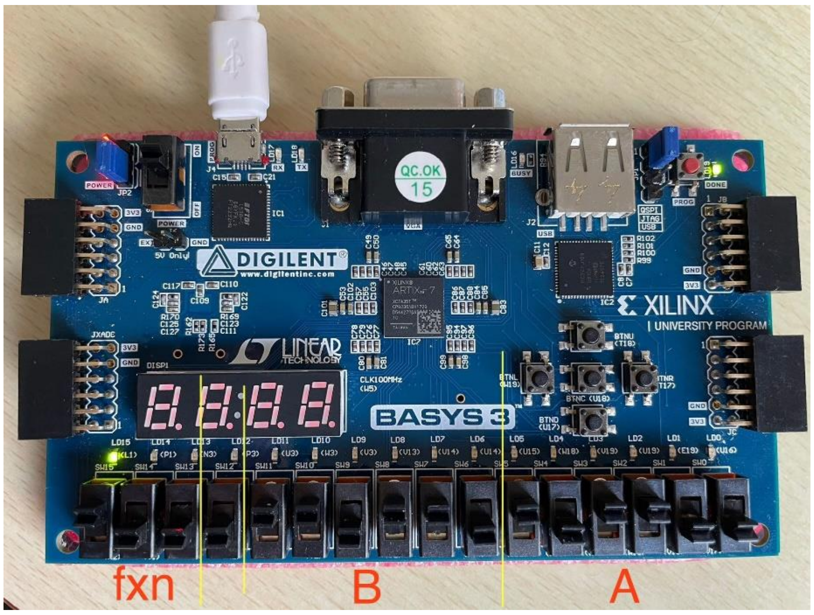


Fig 7.6 fxn = 3'b100 output on Basys 3 Board

Here we observe that the value we give to B and the value we give to A both are being compared and then we see the output on board. Like we have given A the decimal value 44 in binary and B the decimal value 54 in binary so both are being compared and since in this case A is less than B, we get the output on the board by the last LED.

Test Case 6: Displaying A XNOR B
fxn = 3’b101
Test Vector A = 6'b101100
Test Vector B = 6'b010101

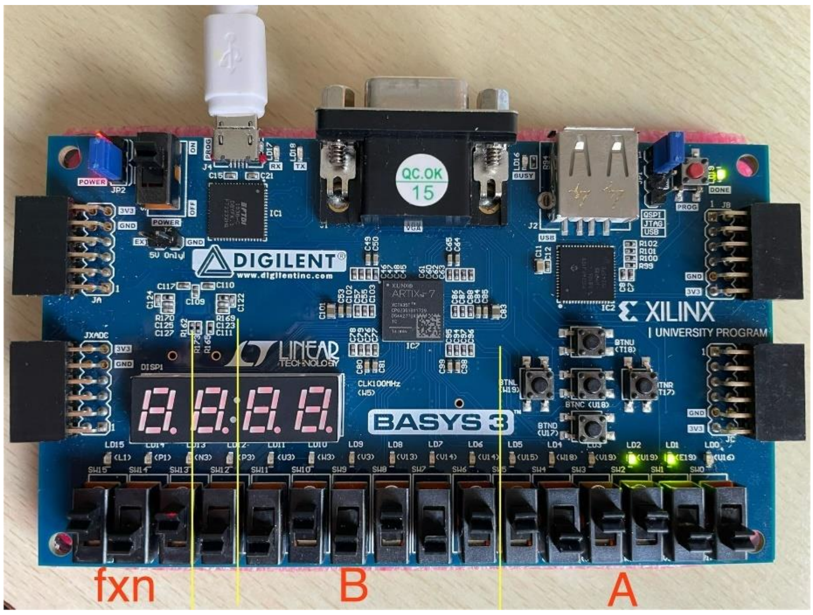


Fig 7.7 fxn = 3’b101 output on Basys 3 Board

Here we have performed xnor operation between A and B and the output is shown on board. Like we have given A the decimal value 44 in binary and B the decimal value 21 in binary, then their logical xnor operation is implemented which gives the output on the board by the first six LEDs.

Test Case 7: Displaying A + B
fxn = 3’b011
Test Vector A = 6'b101100
Test Vector B = 6'b001000
Sel = 1’b0

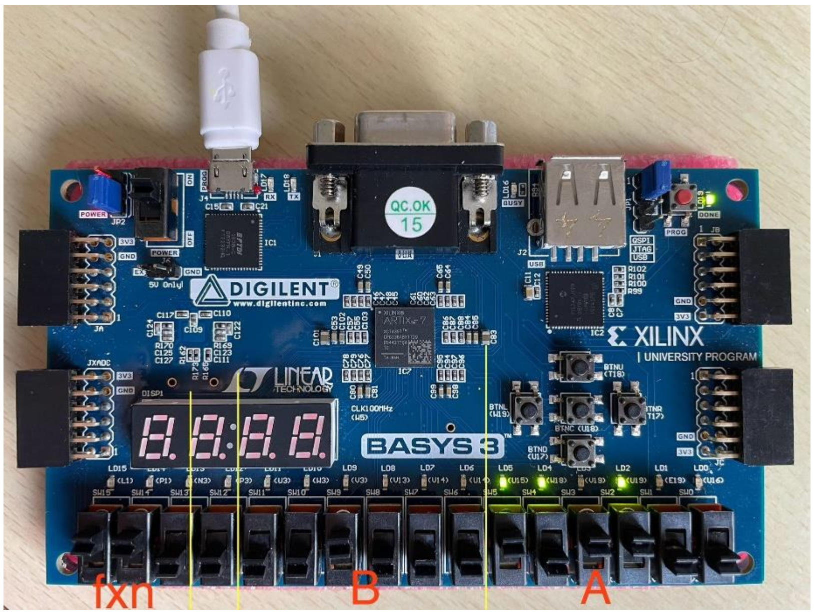


Fig 7.8 fxn = 3’b110 output on Basys 3 Board

Here we have performed addition operation between A and B and the output is shown on board. Like we have given A the decimal value 44 in binary and B the decimal value 8 in binary, then their logical addition operation is implemented which gives the output on the board by the first six LEDs.

<u>Test Case 8: Displaying A - B</u>
fxn = 3'b011
Test Vector A = 6'b101100
Test Vector B = 6'b001000
Sel = 1'b1

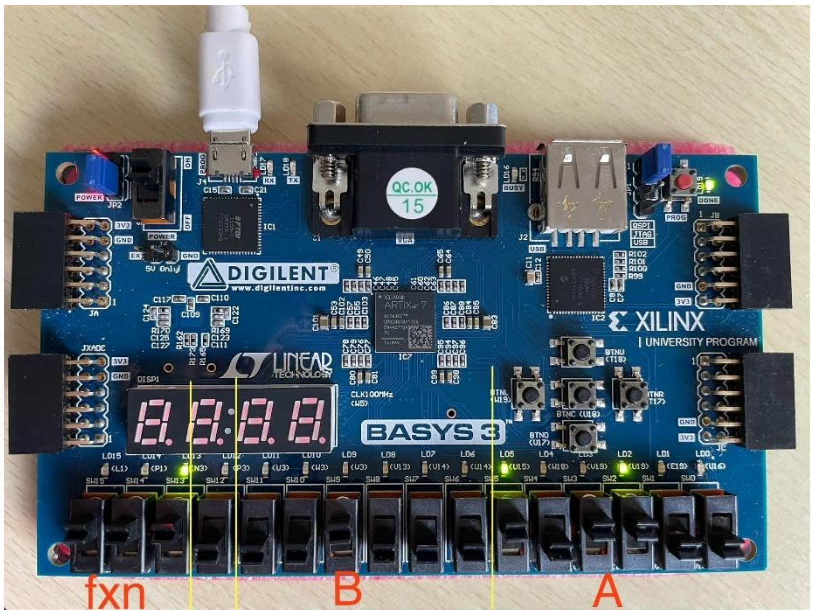
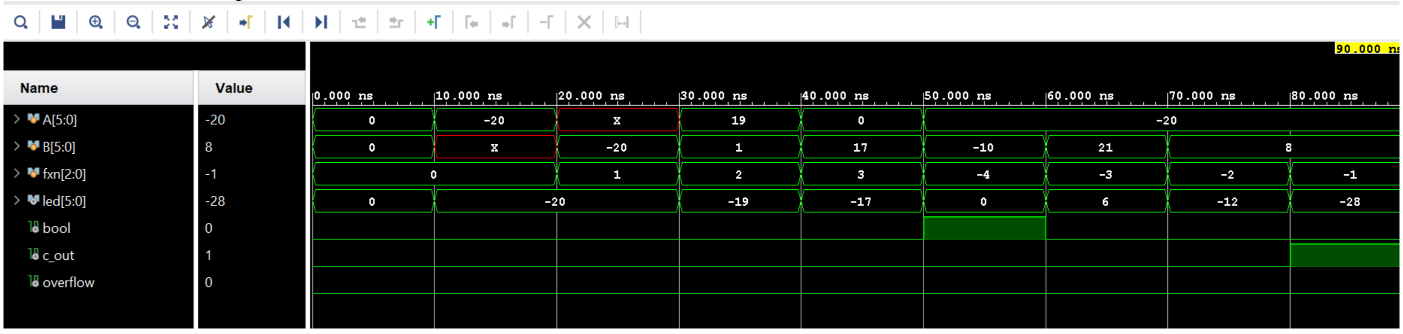


Fig 7.9 fxn = 3'b111 output on Basys 3 Board

Here we have performed subtraction operation between A and B and the output is shown on board. Like we have given A the decimal value 44 in binary and B the decimal value 8 in binary, then their logical subtraction operation is implemented which gives the output on the board by the first six LEDs and since a carry is generated, we also see the third to last LED on.

We also designed a testbench module to test these cases using the vectors and thus verify that if the outputs on board are correct or not.



Name	Value
A[5:0]	-20
B[5:0]	8
fxn[2:0]	-1
led[5:0]	-28
bool	0
c_out	1
overflow	0

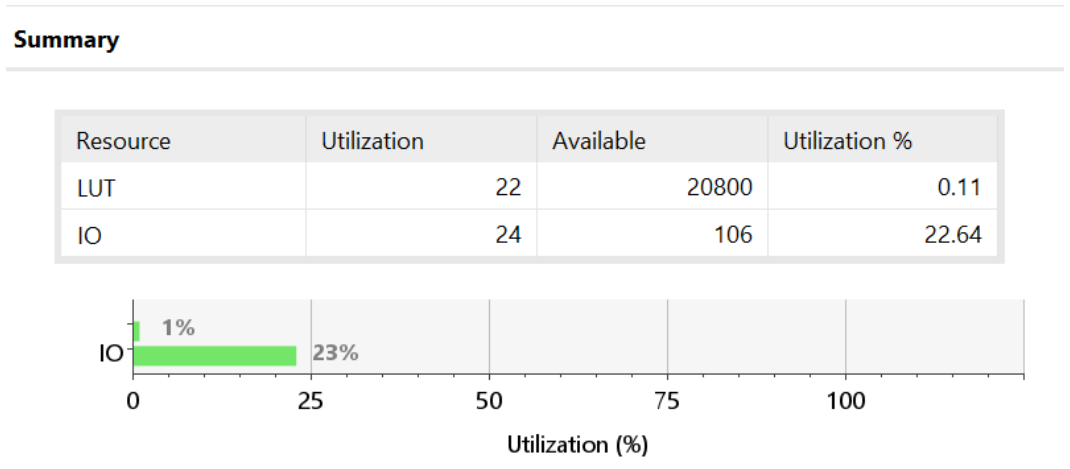
```
90.000 ns
0.000 ns   10.000 ns   20.000 ns   30.000 ns   40.000 ns   50.000 ns   60.000 ns   70.000 ns   80.000 ns
A[5:0]     0    -20    x    19    0    -20
B[5:0]     0    x    -20    1    17    -10    21    8
fxn[2:0]   0    1    2    3    -4    -3    -2    -1
led[5:0]   0    -20    -19    -17    0    6    -12    -28
```

Fig 7.10 Behavioural simulation output on Vivado

Here we are testing various logical and arithmetic operation by using test vectors. A[5:0] & B [5:0] are 6-bit input numbers and fxn[2:0] is the 3-bit input to select the case. For output we have led[5:0], bool, c_out, overflow. We have taken 1 extra input of all zeros to initialize then we have tested for each case. The x signal marked inside red box means no specific input specified. Thus, we verify using this behavioural simulation that all are test cases match as it were on the board.

8. Observations:

Utilization Report, which is produced following the synthesis and implementation phases, provides information on how well our design is utilizing the FPGA's resources. Here we observe that IO resource is utilizing the most percentage of resources, i.e., 23% .

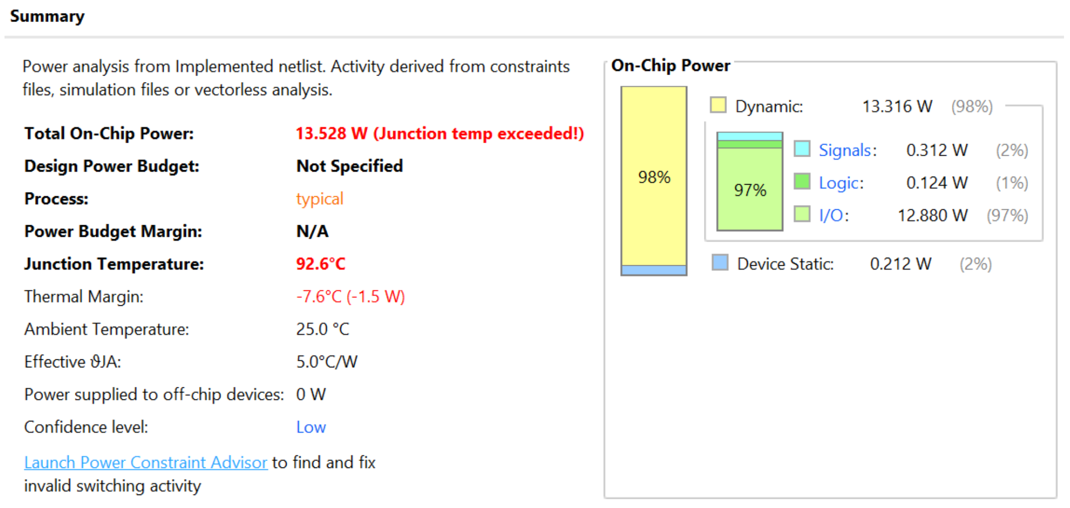


Summary

Resource	Utilization	Available	Utilization %
LUT	22	20800	0.11
IO	24	106	22.64

Fig 8.1 Summary of Utilization Report

Power Consumption Report provides information about the target FPGA device's power consumption profile throughout the execution of our design. Here we observe that the total on-chip power is 13.528W and the Junction Temperature is 92.6℃ which tells the designer reliability of their FPGA designs and making thermal management strategies to ensure safe operation of device. Here we see that the usage of chip exceeds the junction temperature thus resulting in Thermal Margin of -7.6℃. The figure on right tells us that which parameter utilizes how much of the chip power like in this case I/O utilizes the most power amongst Signals, Logic, and I/O.



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	13.528 W (Junction temp exceeded!)
Design Power Budget:	Not Specified
Process:	typical
Power Budget Margin:	N/A
Junction Temperature:	92.6°C
Thermal Margin:	-7.6°C (-1.5 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

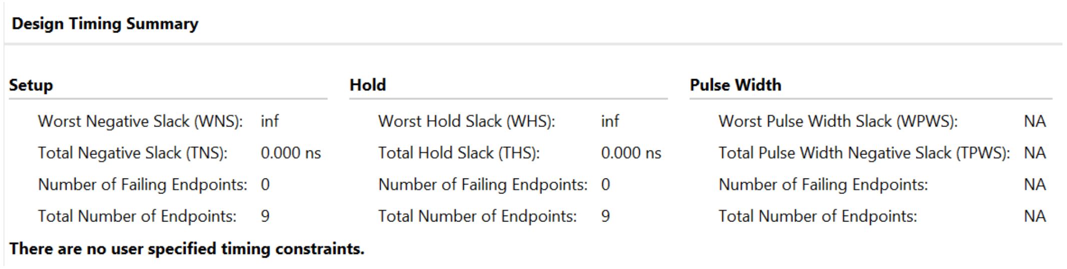
Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	13.316 W	(98%)
Signals:	0.312 W	(2%)
Logic:	0.124 W	(1%)
I/O:	12.880 W	(97%)
Device Static:	0.212 W	(2%)

Fig 8.2 Summary of Power Report

Design Timing Report provides insights into our design's timing behaviour, including clock limitations, timing violations, and key routes. For a designer, this is the most crucial report because it is only via analysis of this report that he can determine whether he is fulfilling his timing goals. Here we see that we do not get any values because we have not specified any timing constraints for this assignment.



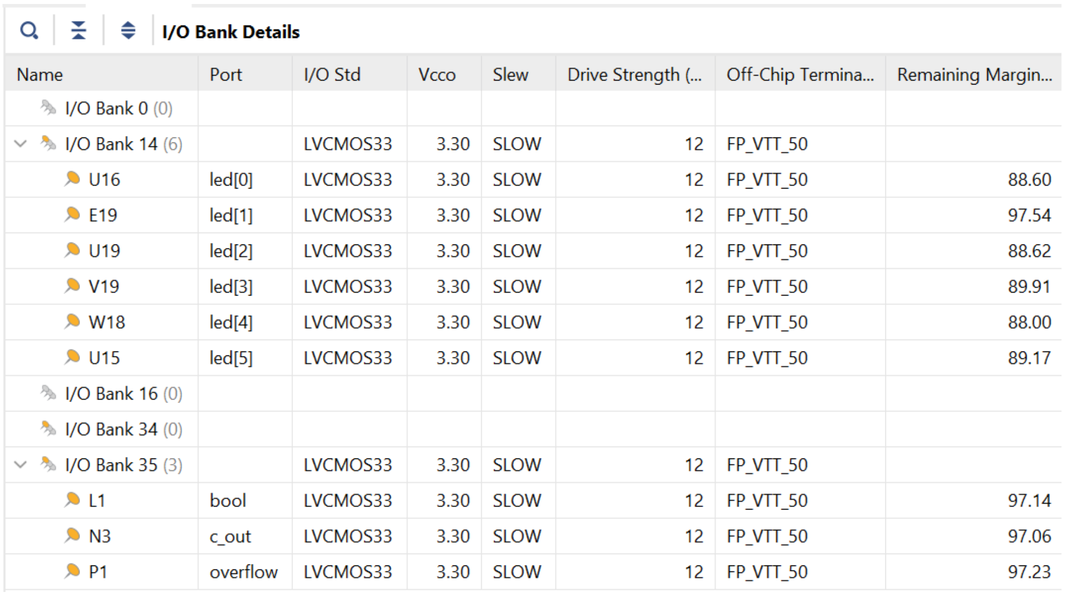
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	9	Total Number of Endpoints:	9	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Fig 8.3 Summary of Design Timing

Noise Report is a useful tool for evaluating the noise properties of our FPGA design, spotting any noise-induced errors and signal integrity problems, and putting mitigation plans into practice to improve design performance and reliability.



I/O Bank Details

Name	Port	I/O Std	Vcco	Slew	Drive Strength (...	Off-Chip Termina...	Remaining Margin...
I/O Bank 0 (0)							
I/O Bank 14 (6)		LVCMOS33	3.30	SLOW	12	FP_VTT_50	
U16	led[0]	LVCMOS33	3.30	SLOW	12	FP_VTT_50	88.60
E19	led[1]	LVCMOS33	3.30	SLOW	12	FP_VTT_50	97.54
U19	led[2]	LVCMOS33	3.30	SLOW	12	FP_VTT_50	88.62
V19	led[3]	LVCMOS33	3.30	SLOW	12	FP_VTT_50	89.91
W18	led[4]	LVCMOS33	3.30	SLOW	12	FP_VTT_50	88.00
U15	led[5]	LVCMOS33	3.30	SLOW	12	FP_VTT_50	89.17
I/O Bank 16 (0)							
I/O Bank 34 (0)							
I/O Bank 35 (3)		LVCMOS33	3.30	SLOW	12	FP_VTT_50	
L1	bool	LVCMOS33	3.30	SLOW	12	FP_VTT_50	97.14
N3	c_out	LVCMOS33	3.30	SLOW	12	FP_VTT_50	97.06
P1	overflow	LVCMOS33	3.30	SLOW	12	FP_VTT_50	97.23

Fig 8.4 Summary of Noise Report

9. Appendix:

9.1 LAB A:

In this lab session we familiarized with Vivado environment and learnt how to find our way through the software for implementing a behavioural simulation. We got to know how to add modules and create new ones to write Verilog language code. We also learnt how to write a testbench module and how we can specify test vectors. In this assignment we used the knowledge to write testbench file and run the behavioural simulation using the test vectors to analyse our logical implementation's.

9.2 LAB B:

In this lab session we were given two modules eq1.v and eq2.v which had the logic of 1-bit comparator and 2-bit comparator respectively. We used these modules and created some new modules to write code for 8-bit greater than logical circuit. In the first new module that we created gt.v, we wrote the logic for greater than circuit and the other new module gt_eq.v, we used the instantiations of these other modules to finally implement 8-bit comparator. In this assignment we used the knowledge to implement the less than logic for two 6-bit inputs by padding the 6-bit input with 2'b00 to correctly instantiate the gt_eq.v module. Then in the topmodule.v file we take the negation of the output to convert it into less than circuit.

9.3 LAB C:

In this lab session we were given the module fulladder.v which had the code for 1-bit full adder. We created two new modules for the testbench and the 6-bit ripple adder/subtractor. In the six_bit_ripple_adder.v module we instantiated the full adder module 6 times and provided inputs and the sel value to accordingly perform addition/subtraction. In this assignment we used this module directly to implement 6 out of 8 operations.

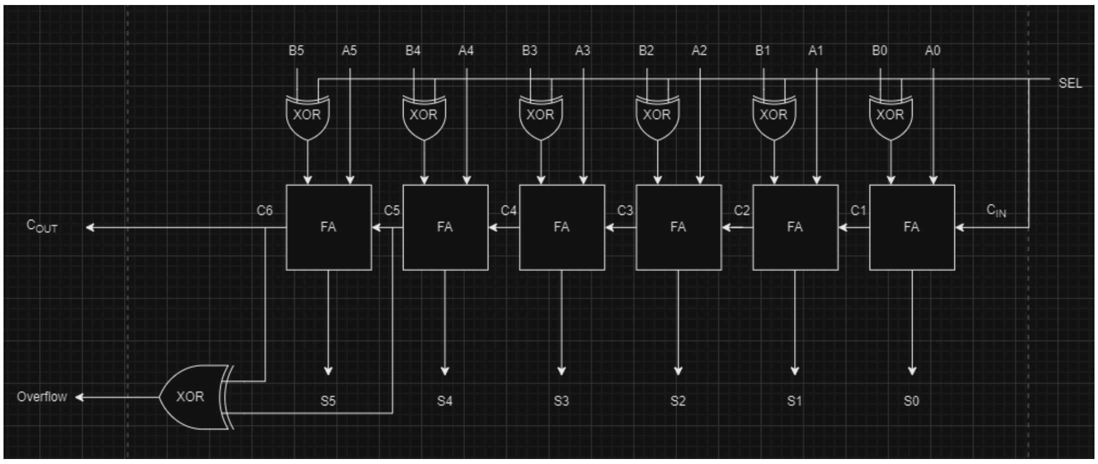


Fig 9.1 6-Bit Full Ripple Adder/Subtractor Diagram

In the above Circuit diagram, we have created a 6-Bit adder/subtractor which basically adds or subtracts two input binary numbers and gives us the result. Additionally, we are also checking the sum of each individual adder and the overflow of the 6-Bit adder/subtractor.

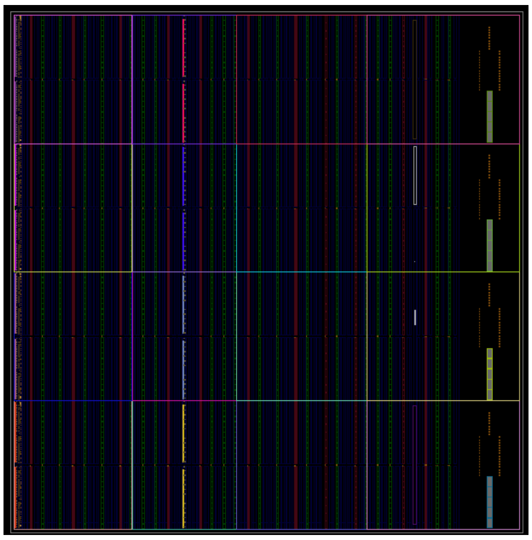


Fig 9.2 Implemented Device for Lab C

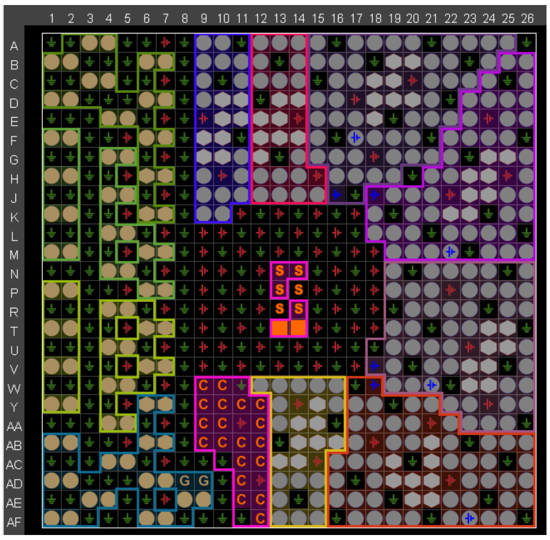


Fig 9.3 Implemented package for Lab C

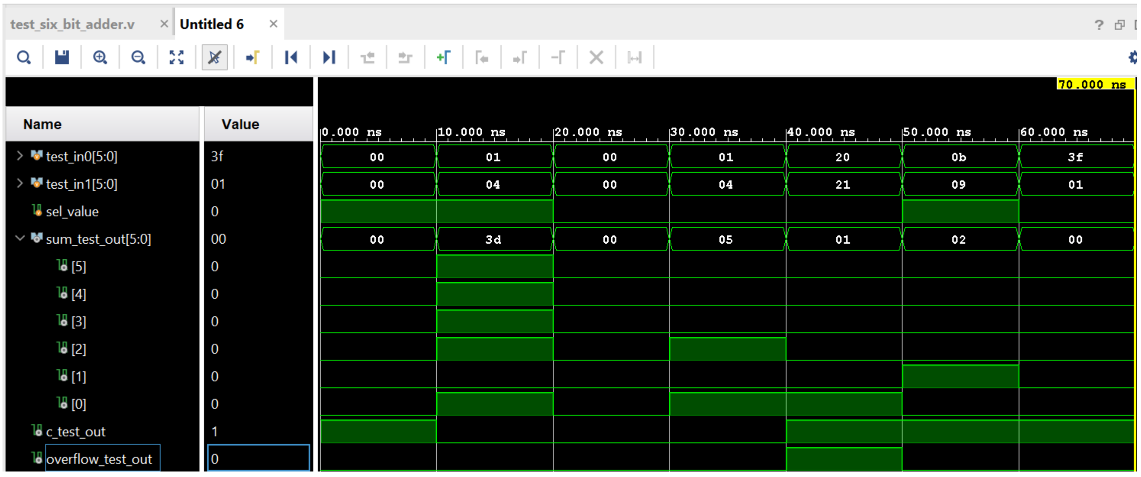


Fig 9.4 Waveform Indicating 7 test cases

As we can see from the above waveform that few cases do not pass. For test case 1 we observe that a carryout is not expected but the waveform has it. For test case 5 we observe that neither of carryout, overflow, and sum are same as observed. For test case 7 the expected sum is not what is observed in the waveform.

9.4 LAB D:

In this lab session we learnt how to use the Basys 3 Board. We observed how switches can be used to give inputs and how the outputs can be seen represented by the LEDs. We updated the constraints file, i.e., xdc file to assign switches as inputs and LEDs as outputs. In this assignment we used the knowledge to correctly update the xdc file and to see how the board works correctly.

9.5 LAB E:

In this lab session we implemented the code we designed in LAB C on the board to give two 6-bit inputs using the switches and the sel value using the switch. We then observed the resultant sum, carry out, and overflow on the board by the LEDs. In this assignment we used all this knowledge to work our way through to correctly use the board.

10. Conclusion:

The 3C7 Digital Systems Design assignment successfully demonstrated the design, testing, and implementation of a mini–Arithmetic Logic Unit (ALU) on the Basys-3 Board, synthesizing knowledge from previous lab experiences. The assignment demonstrated the capacity to apply logical processes, combine many components, and effectively use the FPGA's resources. Through this exercise, we got hands-on learning, reinforcing our understanding of digital systems design principles. The process of designing a comprehensive testbench, analysing waveforms, and ensuring the functionality of the ALU on hardware provided invaluable practical experience. The generated reports provided insights into the performance of the design, identifying opportunities for future improvements and optimization. These reports included the usage, power consumption, design timing, and noise reports.

11. References:

- Lecture Slides
- Lab materials provided on blackboard
- Basic knowledge of electronics from previous modules
- Doubt clearing from the demonstrators