C

3C7 DIGITAL SYSTEMS DESIGN LABORATORY

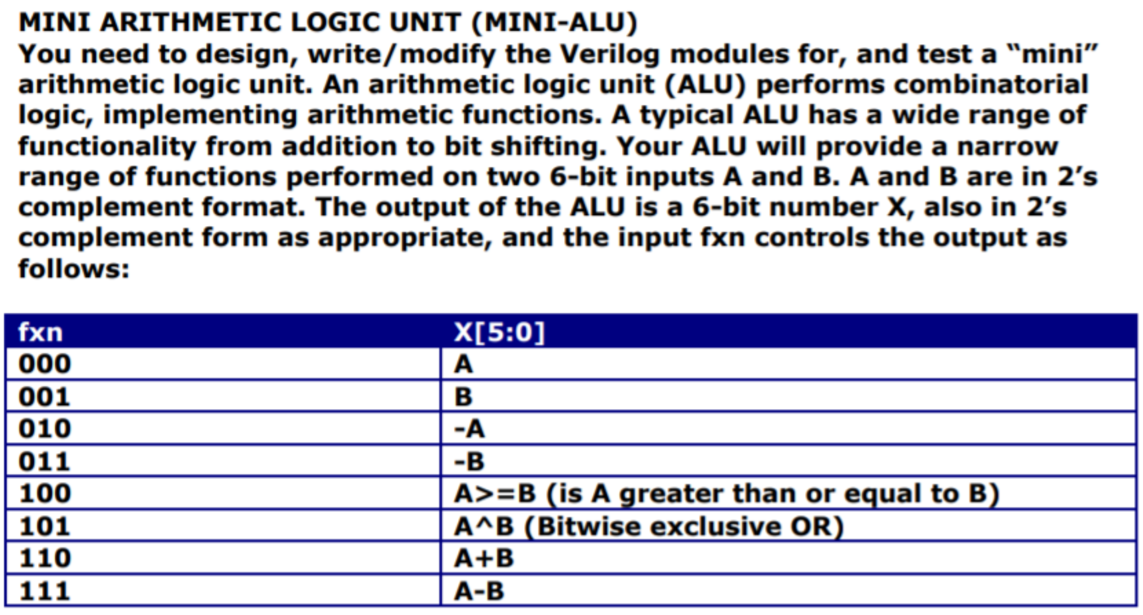
Assignment 1

Department of Electronic and Electrical Engineering

(e-Report submission)

Description:

- Implement Mini ALU using previous lab elements and test it on Basys3 board (no. 60).



MINI ARITHMETIC LOGIC UNIT (MINI-ALU)
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

Submitted by:

Rohan Taneja
19323238
Submission Date: 12/03/2020

1. Implementation

The implementation of Mini Arithmetic Logical Unit onto the Basys3 required initialization with multiple modules. Taken from previous Labs, the module such as eq1, eq2, 6bit_ripple_adder, and full_adder. Now, I implemented the design further with addition of two more modules not6 (sub module not1 – for singular bit), and xor6 (sub module xor1 – for singular bit).

2. Simulations:

This section is divided in into further eight cases of functional selection of ALU to operate as defined is description taken from Assignment 1.

The test vector used are as follows:

```
//test vector 1
A = 6'b000100;
B = 6'b001000;
# 100;
//test vector 2
A = 6'b000110;
B = 6'b010001;
# 100;
```

```
//test vector 3
A = 6'b110010;
B = 6'b000000;
# 100;
//test vector 4
A = 6'b010010;
B = 6'b001000;
# 100;
```

```
//test vector 5
A = 6'b001000;
B = 6'b000110;
# 100;
//test vector 6
A = 6'b011000;
B = 6'b000100;
# 100;
```

Case 0: **fxn = b'000**

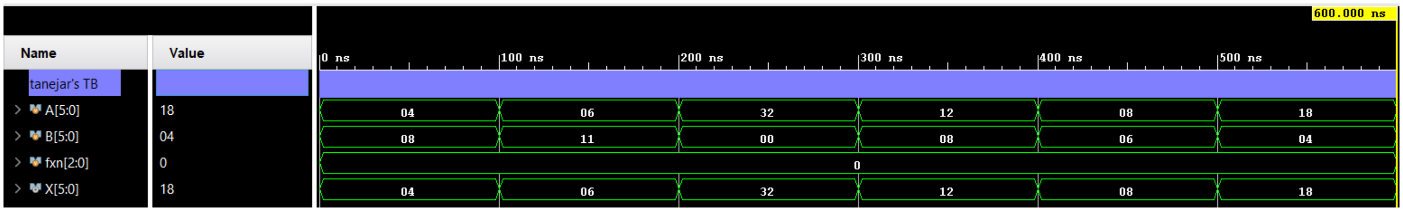

600.000 ns
Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
tanejar's TB
A[5:0] 18 04 06 32 12 08 18
B[5:0] 04 08 11 00 08 06 04
fxn[2:0] 0 0
X[5:0] 18 04 06 32 12 08 18

Fig 2.0 – display A waveform

Case 1: **fxn = b'001**

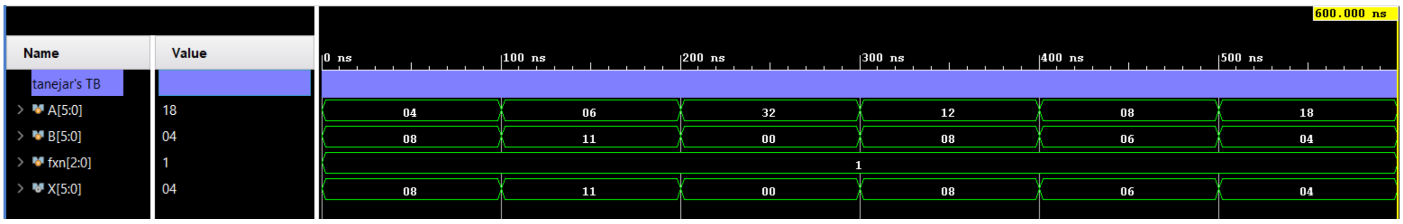

600.000 ns
Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
tanejar's TB
A[5:0] 18 04 06 32 12 08 18
B[5:0] 04 08 11 00 08 06 04
fxn[2:0] 1 1
X[5:0] 04 08 11 00 08 06 04

Fig 2.1 – display B waveform

Case 2: **fxn = b'010**

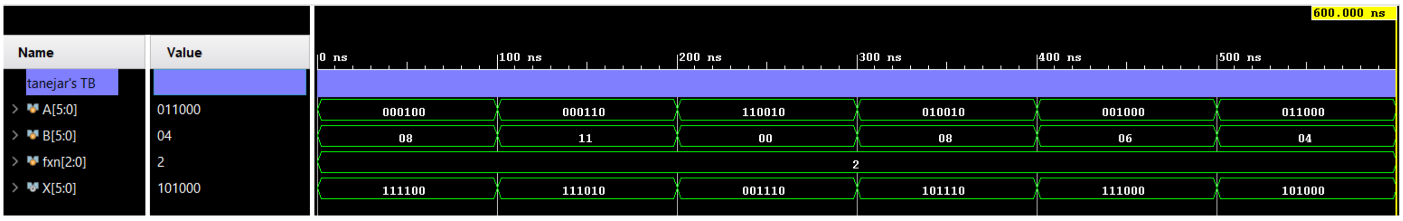

600.000 ns
Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
tanejar's TB
A[5:0] 011000 000100 000110 110010 010010 001000 011000
B[5:0] 04 08 11 00 08 06 04
fxn[2:0] 2 2
X[5:0] 101000 111100 111010 001110 101110 111000 101000

Fig 2.2 – display -A (2's compliment) waveform – radix set to binary

Case 3: **fxn = b’011**

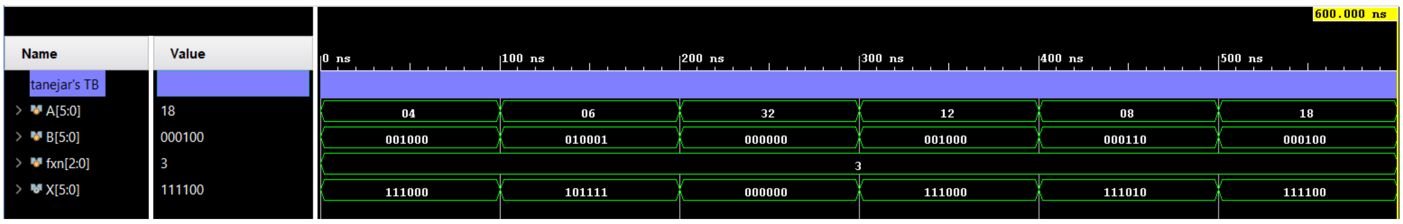


Fig 2.3 – display -B (2’s compliment) waveform – radix set to binary

Case 4: **fxn = b’100**

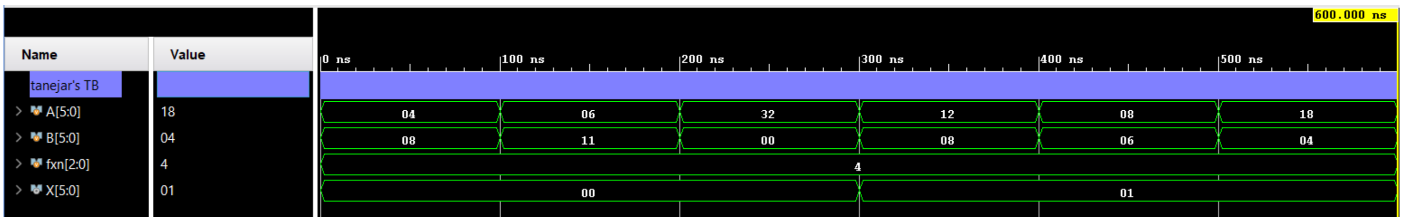


Fig 2.4 – display A>=B waveform

Case 5: **fxn = b’101**

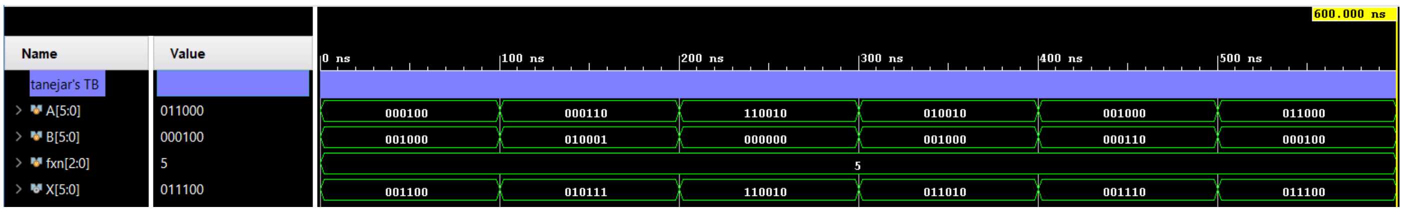


Fig 2.5 – display A^B (bitwise XOR) waveform – radix set to binary

Case 6: **fxn = b’110**

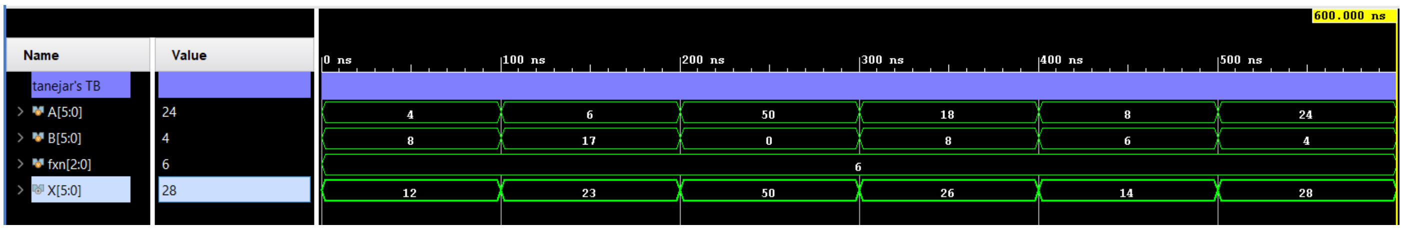


Fig 2.6 – display A+B waveform – radix set to unsigned decimal

Case 7: **fxn = b’111**

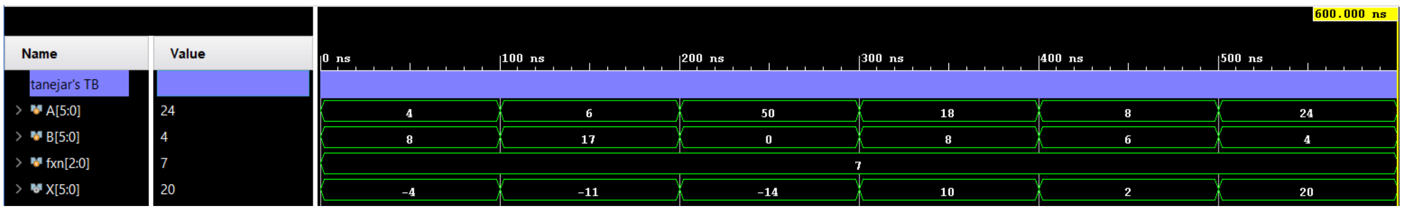


Fig 2.7 – display A-B waveform – radix set to signed decimal

3. Sources:

The project created as **assign1_tanejar** has the following components:

The Design source consist of mini_alu module which inherits properties of other instantiations of modules such as

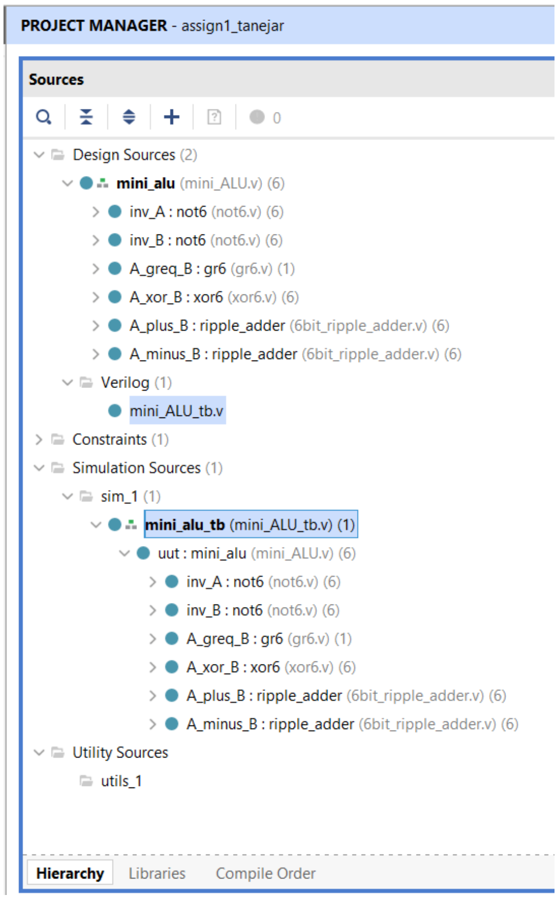
- inv_A,
- inv_B,
- A_greq_B,
- A_xor_B,
- A_plus_B,
- A_minus_B.

The expanded testbench is the simulation source. Which is tested on the Vivado software and represented with waveform in the simulations.

The expanded design sources are the implemented modules used in the project consisting of all the modules mentioned and the inherited modules with them.

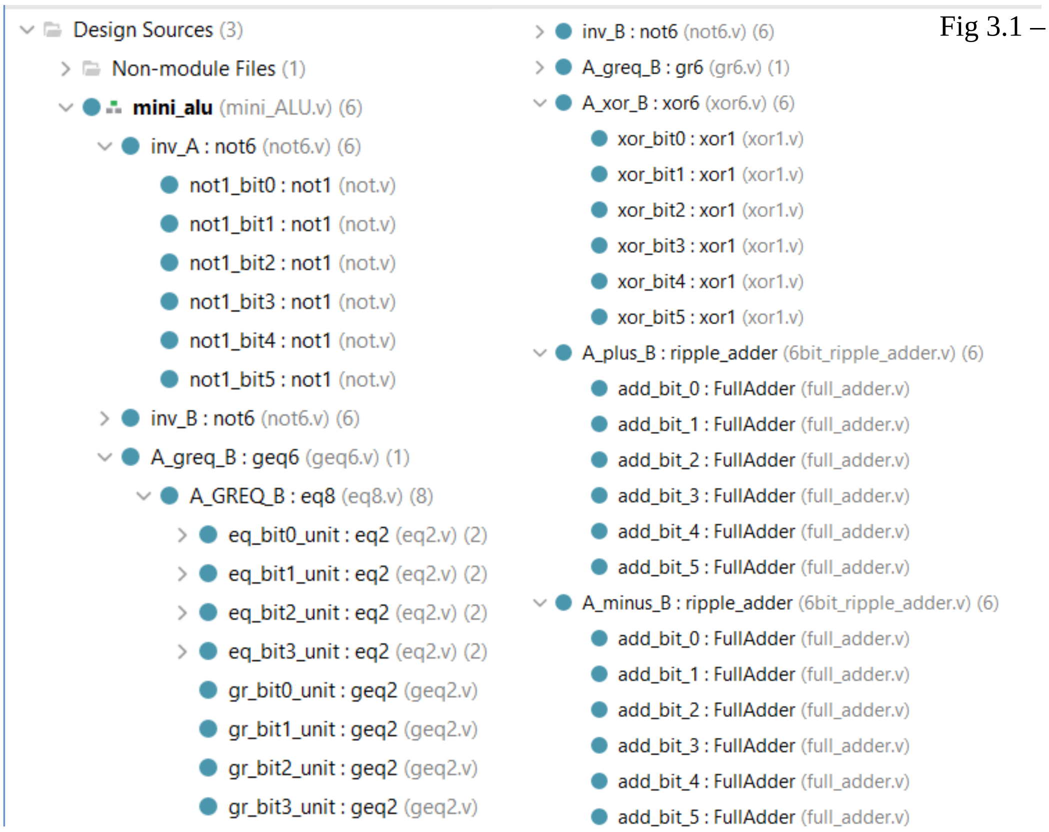
The Testbench is set to simulation only to prevent error while compiling the implementation and bitstream generation.

The file directory structure is mentioned in the next page.



```
PROJECT MANAGER - assign1_tanejar
Sources
Design Sources (2)
  mini_alu (mini_ALU.v) (6)
    inv_A : not6 (not6.v) (6)
    inv_B : not6 (not6.v) (6)
    A_greq_B : gr6 (gr6.v) (1)
    A_xor_B : xor6 (xor6.v) (6)
    A_plus_B : ripple_adder (6bit_ripple_adder.v) (6)
    A_minus_B : ripple_adder (6bit_ripple_adder.v) (6)
  Verilog (1)
    mini_ALU_tb.v
Constraints (1)
Simulation Sources (1)
  sim_1 (1)
    mini_alu_tb (mini_ALU_tb.v) (1)
      uut : mini_alu (mini_ALU.v) (6)
        inv_A : not6 (not6.v) (6)
        inv_B : not6 (not6.v) (6)
        A_greq_B : gr6 (gr6.v) (1)
        A_xor_B : xor6 (xor6.v) (6)
        A_plus_B : ripple_adder (6bit_ripple_adder.v) (6)
        A_minus_B : ripple_adder (6bit_ripple_adder.v) (6)
Utility Sources
  utils_1
Hierarchy  Libraries  Compile Order
```

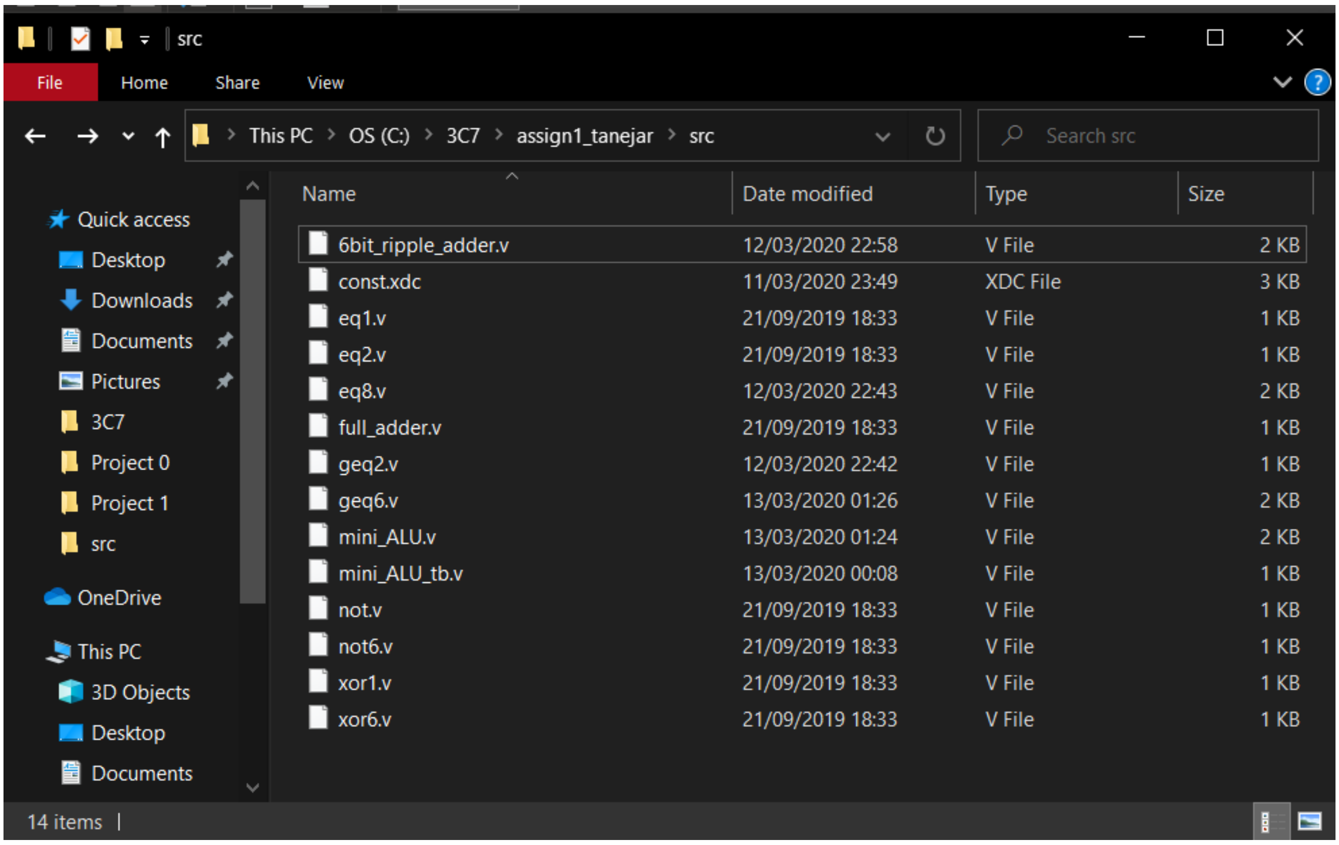
Fig 3.1 – Project Manager - Sources



```
Design Sources (3)
  Non-module Files (1)
  mini_alu (mini_ALU.v) (6)
    inv_A : not6 (not6.v) (6)
      not1_bit0 : not1 (not.v)
      not1_bit1 : not1 (not.v)
      not1_bit2 : not1 (not.v)
      not1_bit3 : not1 (not.v)
      not1_bit4 : not1 (not.v)
      not1_bit5 : not1 (not.v)
    inv_B : not6 (not6.v) (6)
    A_greq_B : geq6 (geq6.v) (1)
      A_GREQ_B : eq8 (eq8.v) (8)
        eq_bit0_unit : eq2 (eq2.v) (2)
        eq_bit1_unit : eq2 (eq2.v) (2)
        eq_bit2_unit : eq2 (eq2.v) (2)
        eq_bit3_unit : eq2 (eq2.v) (2)
        gr_bit0_unit : geq2 (geq2.v)
        gr_bit1_unit : geq2 (geq2.v)
        gr_bit2_unit : geq2 (geq2.v)
        gr_bit3_unit : geq2 (geq2.v)
    inv_B : not6 (not6.v) (6)
    A_greq_B : gr6 (gr6.v) (1)
    A_xor_B : xor6 (xor6.v) (6)
      xor_bit0 : xor1 (xor1.v)
      xor_bit1 : xor1 (xor1.v)
      xor_bit2 : xor1 (xor1.v)
      xor_bit3 : xor1 (xor1.v)
      xor_bit4 : xor1 (xor1.v)
      xor_bit5 : xor1 (xor1.v)
    A_plus_B : ripple_adder (6bit_ripple_adder.v) (6)
      add_bit_0 : FullAdder (full_adder.v)
      add_bit_1 : FullAdder (full_adder.v)
      add_bit_2 : FullAdder (full_adder.v)
      add_bit_3 : FullAdder (full_adder.v)
      add_bit_4 : FullAdder (full_adder.v)
      add_bit_5 : FullAdder (full_adder.v)
    A_minus_B : ripple_adder (6bit_ripple_adder.v) (6)
      add_bit_0 : FullAdder (full_adder.v)
      add_bit_1 : FullAdder (full_adder.v)
      add_bit_2 : FullAdder (full_adder.v)
      add_bit_3 : FullAdder (full_adder.v)
      add_bit_4 : FullAdder (full_adder.v)
      add_bit_5 : FullAdder (full_adder.v)
```

Fig 3.2 – Project – Design Sources

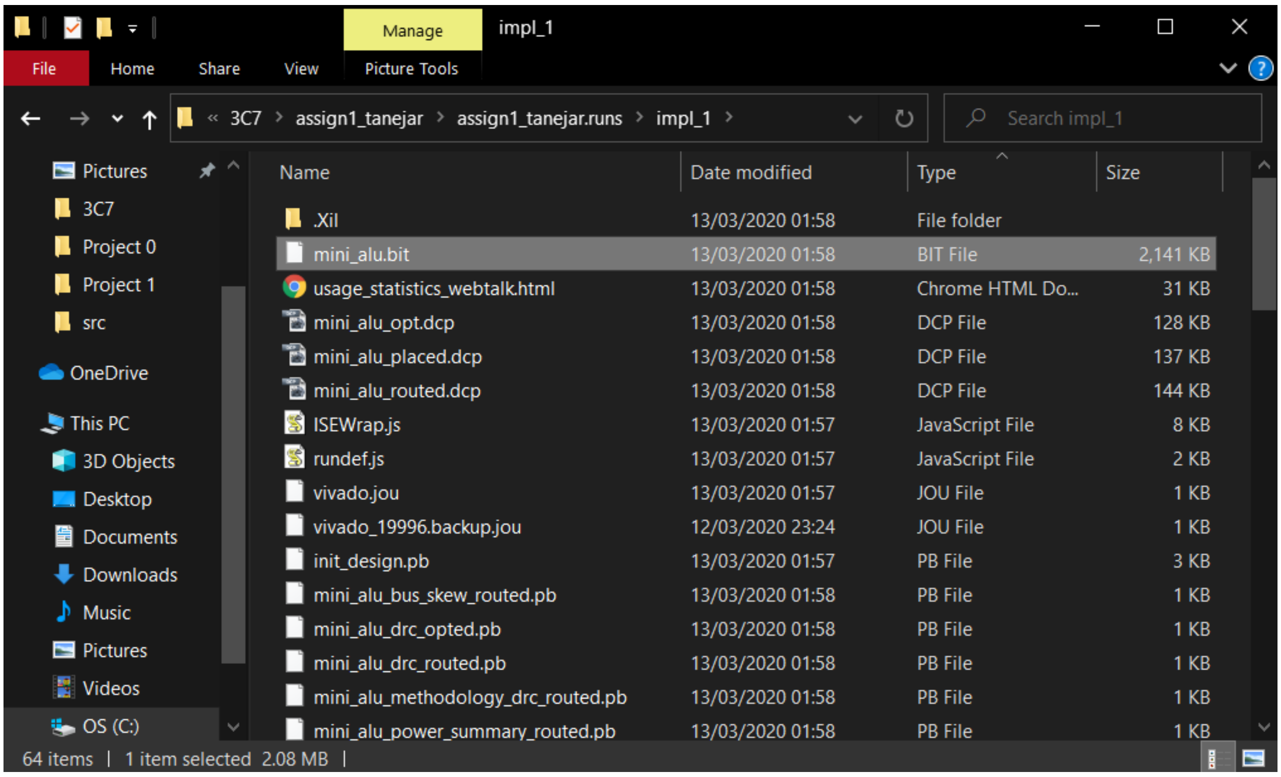
The file directory used is as follows:



Name	Date modified	Type	Size
6bit_ripple_adder.v	12/03/2020 22:58	V File	2 KB
const.xdc	11/03/2020 23:49	XDC File	3 KB
eq1.v	21/09/2019 18:33	V File	1 KB
eq2.v	21/09/2019 18:33	V File	1 KB
eq8.v	12/03/2020 22:43	V File	2 KB
full_adder.v	21/09/2019 18:33	V File	1 KB
geq2.v	12/03/2020 22:42	V File	1 KB
geq6.v	13/03/2020 01:26	V File	2 KB
mini_ALU.v	13/03/2020 01:24	V File	2 KB
mini_ALU_tb.v	13/03/2020 00:08	V File	1 KB
not.v	21/09/2019 18:33	V File	1 KB
not6.v	21/09/2019 18:33	V File	1 KB
xor1.v	21/09/2019 18:33	V File	1 KB
xor6.v	21/09/2019 18:33	V File	1 KB

Fig 3.3 – File Directory with the source files used

This constitutes every module used to generate the bitstream. Previously implemented geq2 from Lab B and 6bit_ripple_adder from Lab C inclusive of provided module was useful in the implementation of this assignment.



Name	Date modified	Type	Size
.Xil	13/03/2020 01:58	File folder	
mini_alu.bit	13/03/2020 01:58	BIT File	2,141 KB
usage_statistics_webtalk.html	13/03/2020 01:58	Chrome HTML Do...	31 KB
mini_alu_opt.dcp	13/03/2020 01:58	DCP File	128 KB
mini_alu_placed.dcp	13/03/2020 01:58	DCP File	137 KB
mini_alu_routed.dcp	13/03/2020 01:58	DCP File	144 KB
ISEWrap.js	13/03/2020 01:57	JavaScript File	8 KB
rundef.js	13/03/2020 01:57	JavaScript File	2 KB
vivado.jou	13/03/2020 01:57	JOU File	1 KB
vivado_19996.backup.jou	12/03/2020 23:24	JOU File	1 KB
init_design.pb	13/03/2020 01:57	PB File	3 KB
mini_alu_bus_skew_routed.pb	13/03/2020 01:58	PB File	1 KB
mini_alu_drc_opted.pb	13/03/2020 01:58	PB File	1 KB
mini_alu_drc_routed.pb	13/03/2020 01:58	PB File	1 KB
mini_alu_methodology_drc_routed.pb	13/03/2020 01:58	PB File	1 KB
mini_alu_power_summary_routed.pb	13/03/2020 01:58	PB File	1 KB

Fig 3.4 – Generated Bitstream for the implementation

4. Schematics Generated:

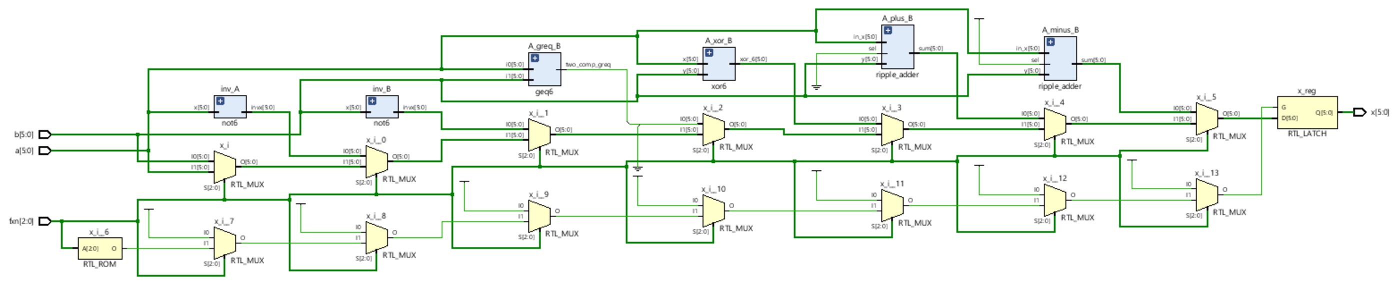


Fig 4.1 - Elaborated Design – generated on assign1_tanejar

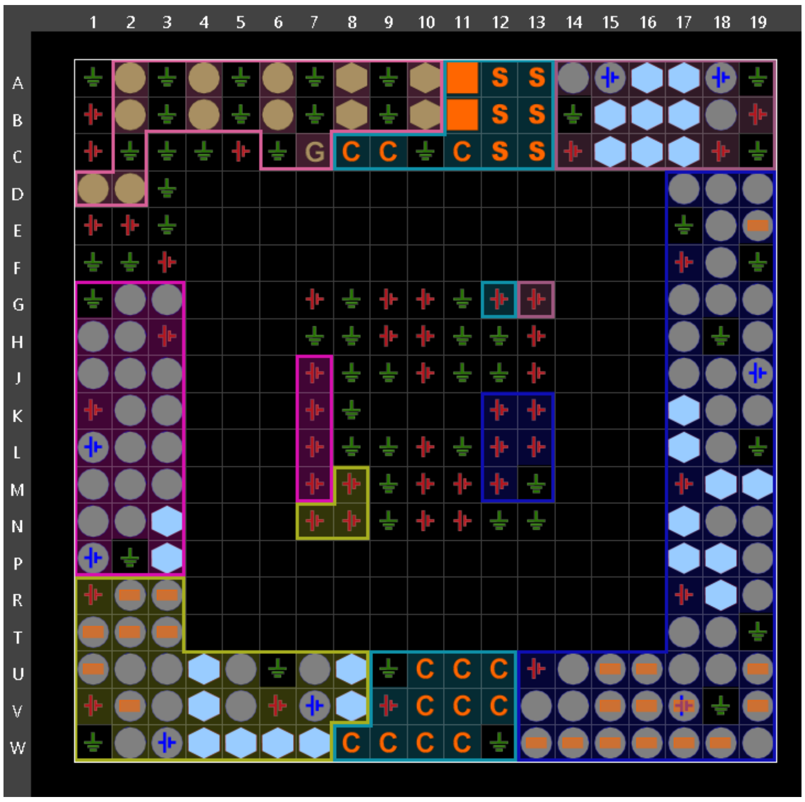


Fig 4.2 – Implemented Package – generated on assign1_tanejar

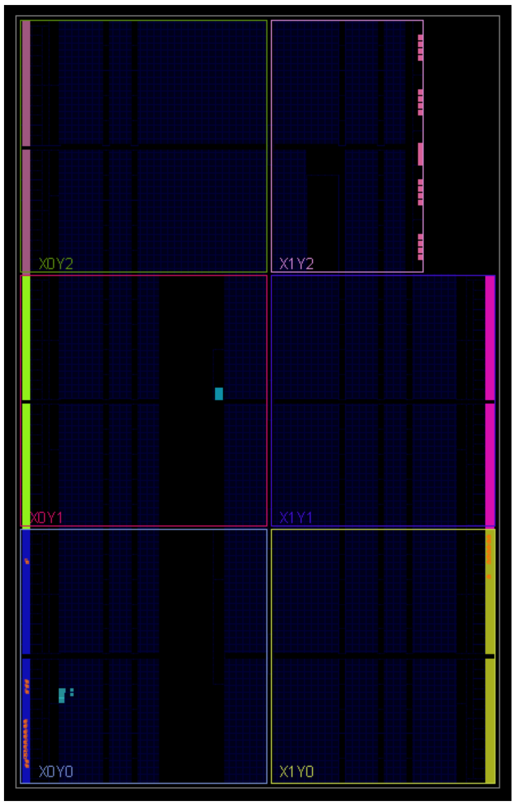


Fig 4.3 – Implemented Device

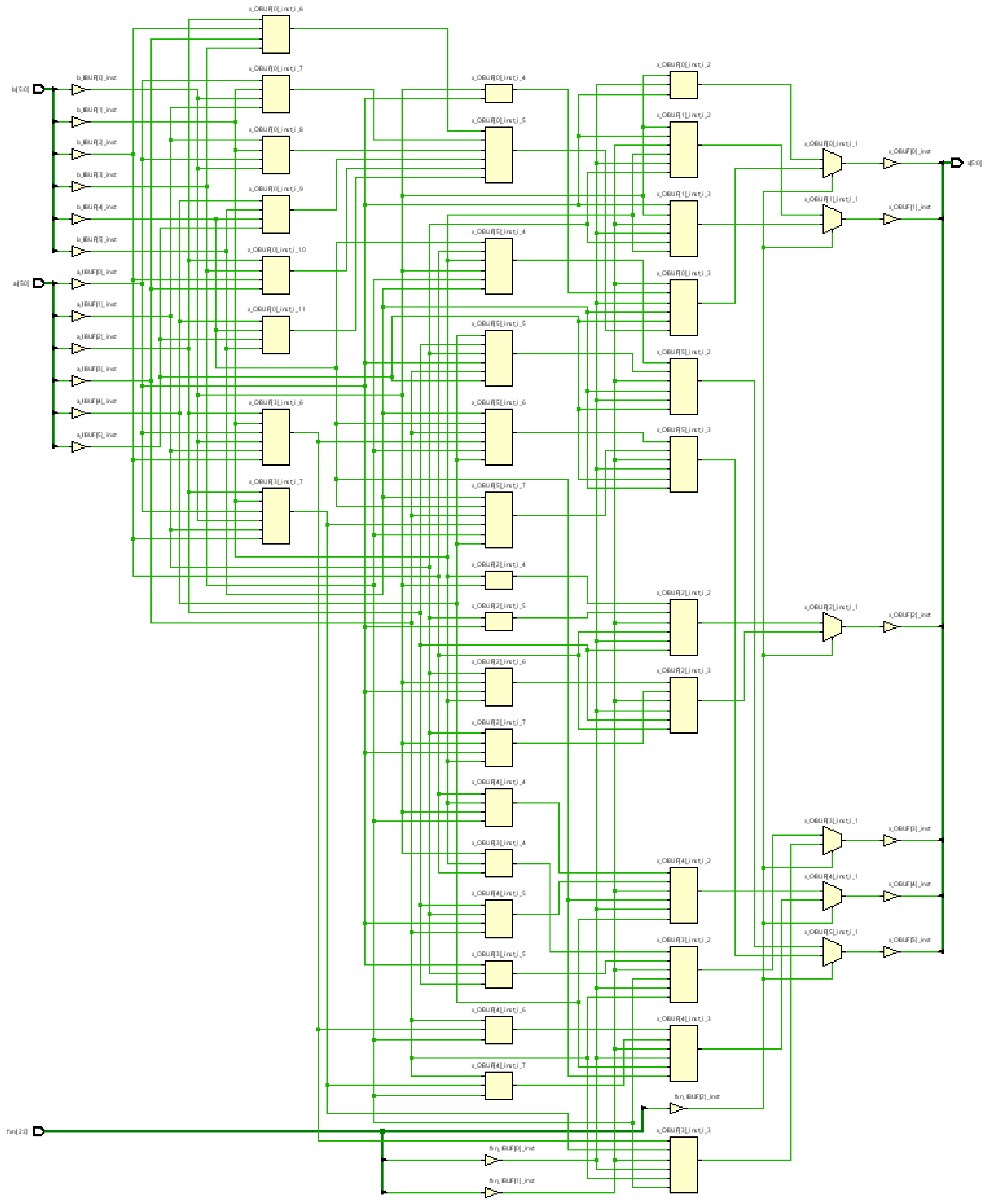
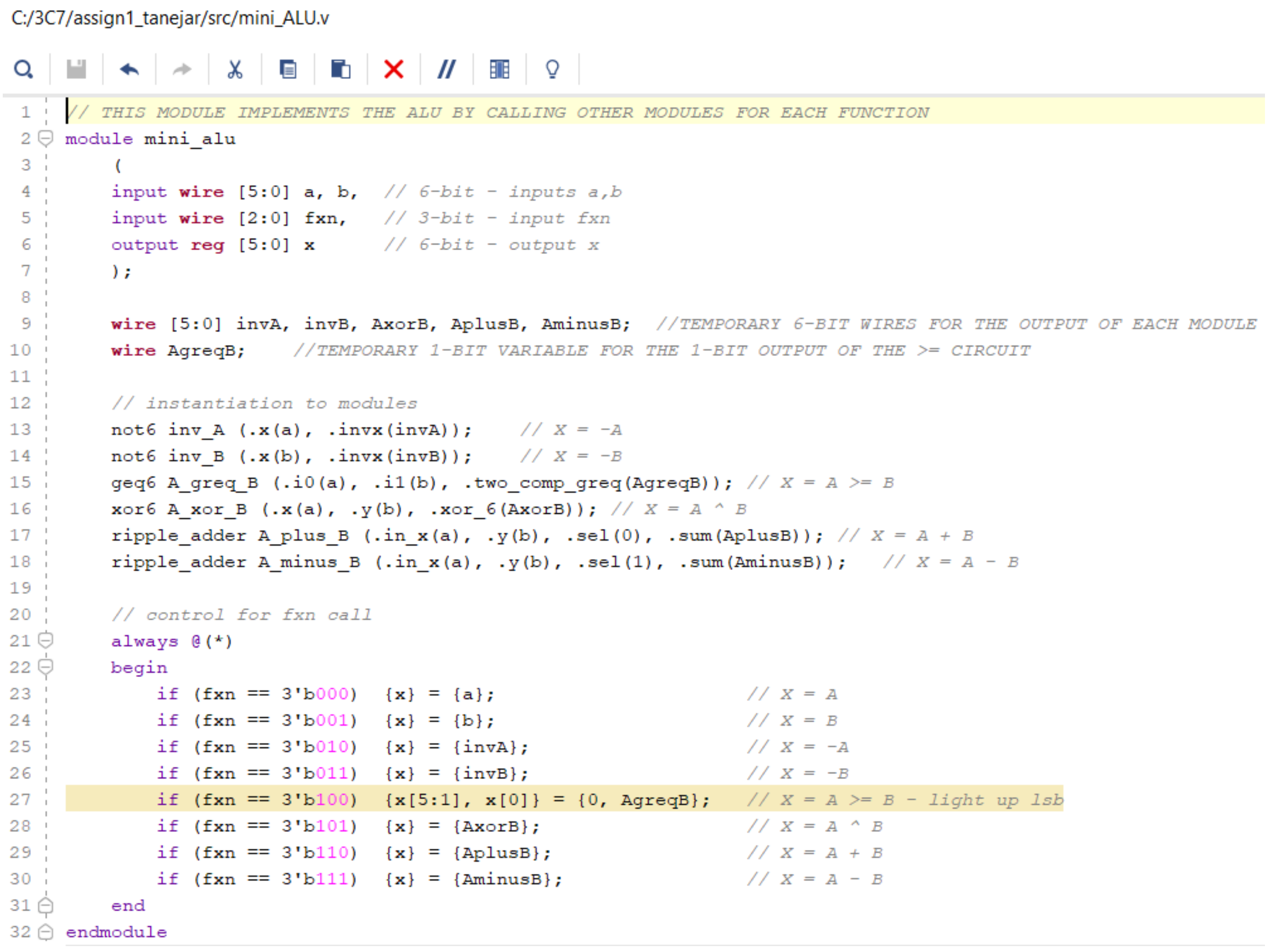


Fig 4.4 – Implemented Schematic – generated on assign1_tanejar

5. Code Snippets:

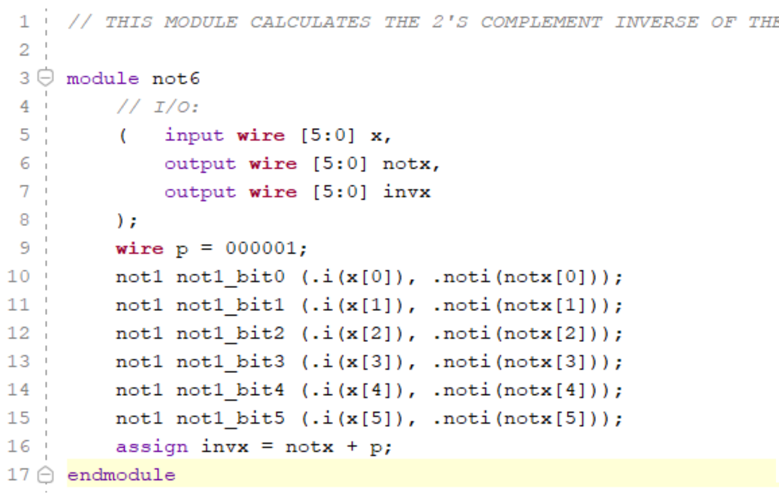
The code snippets to new modules created other than ones from previous Lab B, and C are as follows-



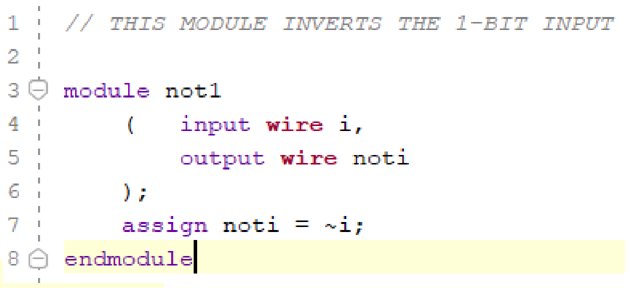
C:/3C7/assign1_tanejar/src/mini_ALU.v

```
// THIS MODULE IMPLEMENTS THE ALU BY CALLING OTHER MODULES FOR EACH FUNCTION
module mini_alu
    (
    input wire [5:0] a, b,   // 6-bit - inputs a,b
    input wire [2:0] fxn,    // 3-bit - input fxn
    output reg [5:0] x       // 6-bit - output x
    );

    wire [5:0] invA, invB, AxorB, AplusB, AminusB;  //TEMPORARY 6-BIT WIRES FOR THE OUTPUT OF EACH MODULE
    wire AgreqB;    //TEMPORARY 1-BIT VARIABLE FOR THE 1-BIT OUTPUT OF THE >= CIRCUIT

    // instantiation to modules
    not6 inv_A (.x(a), .invx(invA));    // X = -A
    not6 inv_B (.x(b), .invx(invB));    // X = -B
    geq6 A_greq_B (.i0(a), .i1(b), .two_comp_greq(AgreqB)); // X = A >= B
    xor6 A_xor_B (.x(a), .y(b), .xor_6(AxorB)); // X = A ^ B
    ripple_adder A_plus_B (.in_x(a), .y(b), .sel(0), .sum(AplusB)); // X = A + B
    ripple_adder A_minus_B (.in_x(a), .y(b), .sel(1), .sum(AminusB));   // X = A - B

    // control for fxn call
    always @(*)
    begin
        if (fxn == 3'b000)  {x} = {a};                      // X = A
        if (fxn == 3'b001)  {x} = {b};                      // X = B
        if (fxn == 3'b010)  {x} = {invA};                   // X = -A
        if (fxn == 3'b011)  {x} = {invB};                   // X = -B
        if (fxn == 3'b100)  {x[5:1], x[0]} = {0, AgreqB};   // X = A >= B - light up lsb
        if (fxn == 3'b101)  {x} = {AxorB};                  // X = A ^ B
        if (fxn == 3'b110)  {x} = {AplusB};                 // X = A + B
        if (fxn == 3'b111)  {x} = {AminusB};                // X = A - B
    end
endmodule
```

Fig 5.1 – mini_ALU module – with instantiation of modules

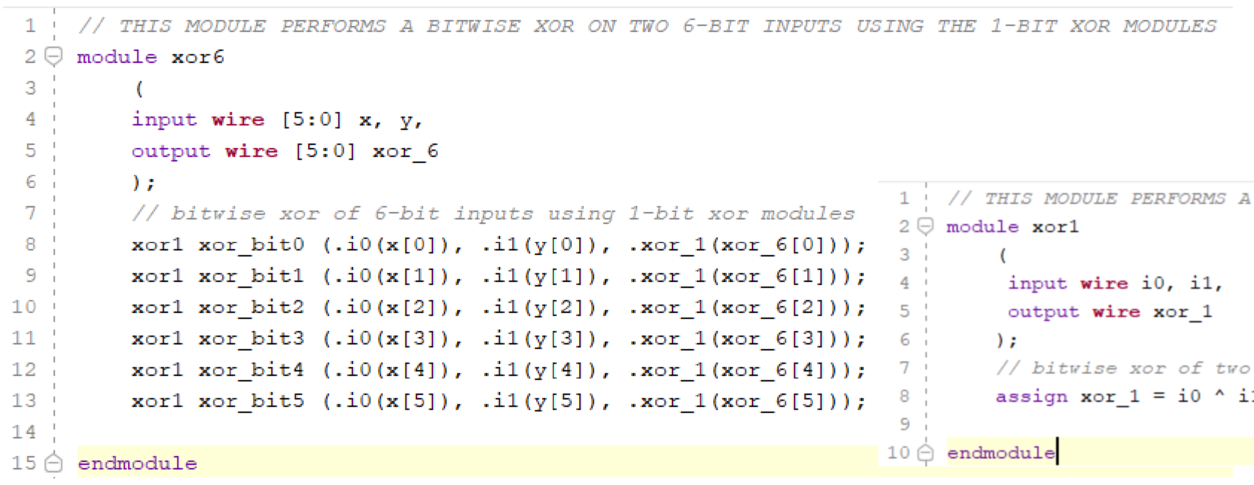


```
// THIS MODULE CALCULATES THE 2'S COMPLEMENT INVERSE OF THE 6-BIT INPUT

module not6
    // I/O:
    (   input wire [5:0] x,
        output wire [5:0] notx,
        output wire [5:0] invx
    );
    wire p = 000001;
    not1 not1_bit0 (.i(x[0]), .noti(notx[0]));
    not1 not1_bit1 (.i(x[1]), .noti(notx[1]));
    not1 not1_bit2 (.i(x[2]), .noti(notx[2]));
    not1 not1_bit3 (.i(x[3]), .noti(notx[3]));
    not1 not1_bit4 (.i(x[4]), .noti(notx[4]));
    not1 not1_bit5 (.i(x[5]), .noti(notx[5]));
    assign invx = notx + p;
endmodule
```

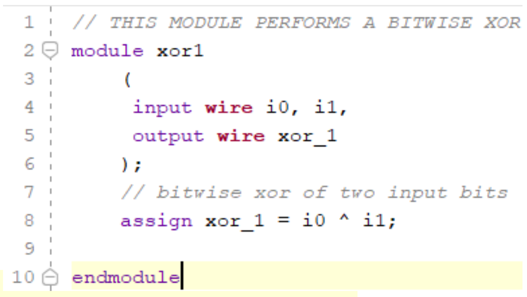


```
// THIS MODULE INVERTS THE 1-BIT INPUT

module not1
    (   input wire i,
        output wire noti
    );
    assign noti = ~i;
endmodule
```

Fig 5.2 – not6 module for 2’s compliment of the input with dependency not1

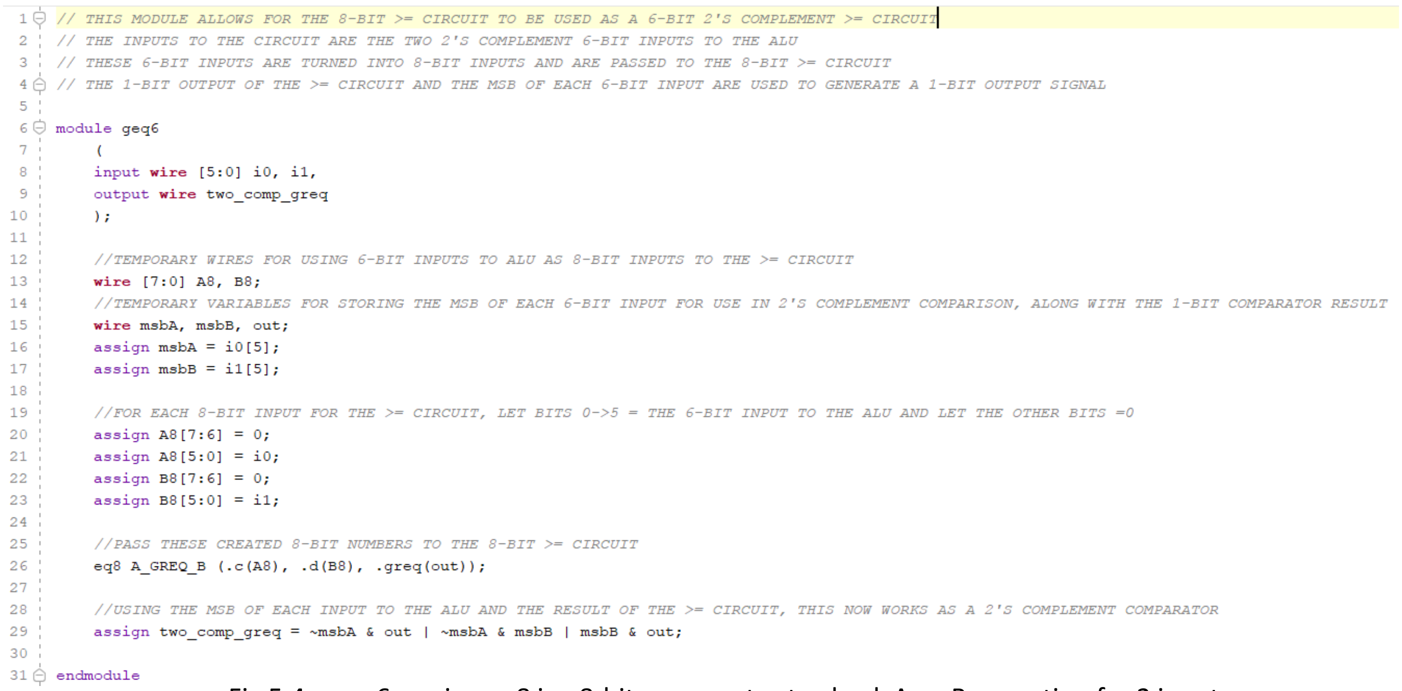


```
// THIS MODULE PERFORMS A BITWISE XOR ON TWO 6-BIT INPUTS USING THE 1-BIT XOR MODULES
module xor6
    (
    input wire [5:0] x, y,
    output wire [5:0] xor_6
    );
    // bitwise xor of 6-bit inputs using 1-bit xor modules
    xor1 xor_bit0 (.i0(x[0]), .i1(y[0]), .xor_1(xor_6[0]));
    xor1 xor_bit1 (.i0(x[1]), .i1(y[1]), .xor_1(xor_6[1]));
    xor1 xor_bit2 (.i0(x[2]), .i1(y[2]), .xor_1(xor_6[2]));
    xor1 xor_bit3 (.i0(x[3]), .i1(y[3]), .xor_1(xor_6[3]));
    xor1 xor_bit4 (.i0(x[4]), .i1(y[4]), .xor_1(xor_6[4]));
    xor1 xor_bit5 (.i0(x[5]), .i1(y[5]), .xor_1(xor_6[5]));

endmodule
```



```
// THIS MODULE PERFORMS A BITWISE XOR
module xor1
    (
     input wire i0, i1,
     output wire xor_1
    );
    // bitwise xor of two input bits
    assign xor_1 = i0 ^ i1;

endmodule
```

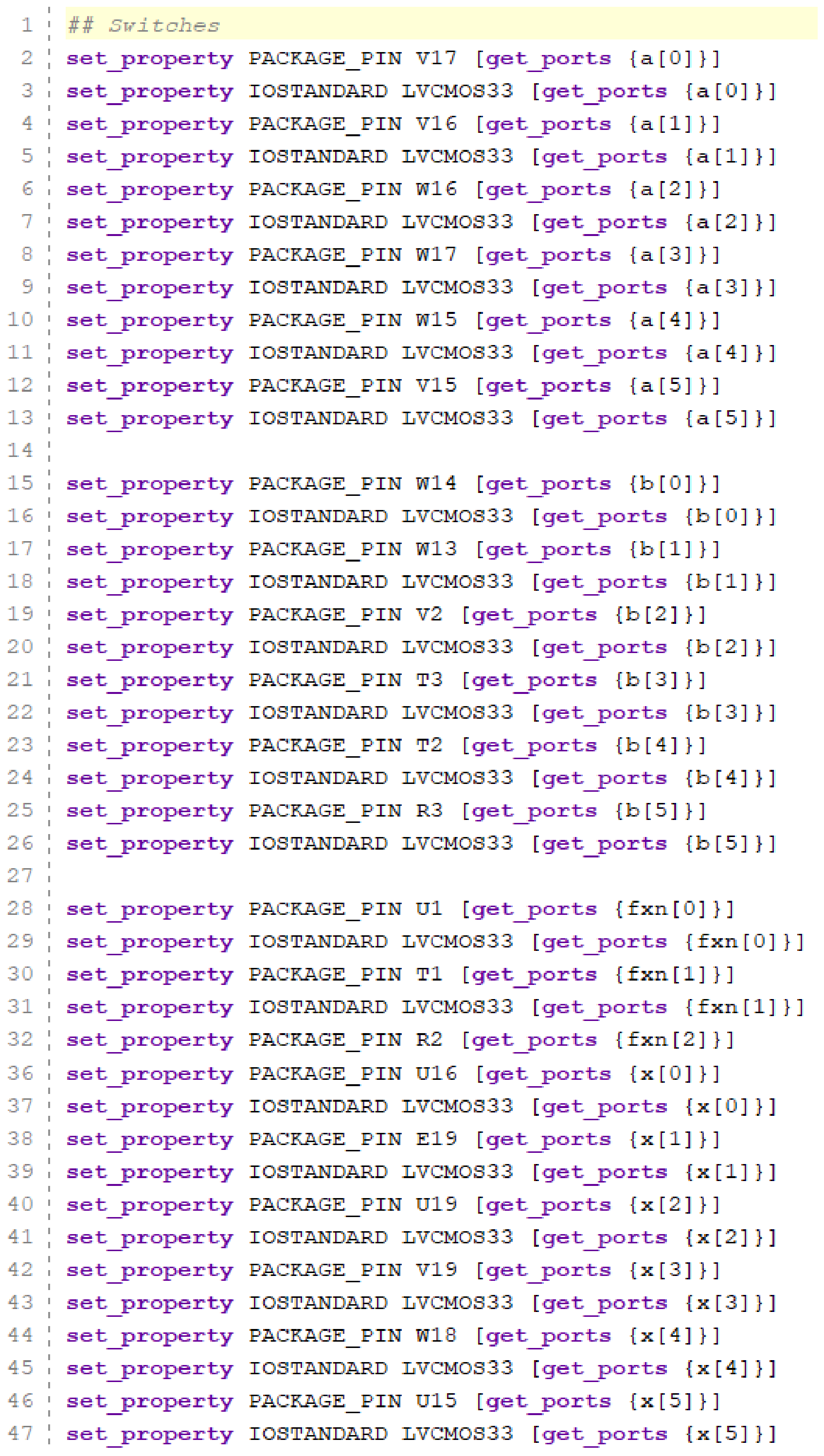
Fig 5.3 – xor6 module for bitwise XOR operation of the input with dependency xor1



```
// THIS MODULE ALLOWS FOR THE 8-BIT >= CIRCUIT TO BE USED AS A 6-BIT 2'S COMPLEMENT >= CIRCUIT
// THE INPUTS TO THE CIRCUIT ARE THE TWO 2'S COMPLEMENT 6-BIT INPUTS TO THE ALU
// THESE 6-BIT INPUTS ARE TURNED INTO 8-BIT INPUTS AND ARE PASSED TO THE 8-BIT >= CIRCUIT
// THE 1-BIT OUTPUT OF THE >= CIRCUIT AND THE MSB OF EACH 6-BIT INPUT ARE USED TO GENERATE A 1-BIT OUTPUT SIGNAL

module geq6
    (
    input wire [5:0] i0, i1,
    output wire two_comp_greq
    );

    //TEMPORARY WIRES FOR USING 6-BIT INPUTS TO ALU AS 8-BIT INPUTS TO THE >= CIRCUIT
    wire [7:0] A8, B8;
    //TEMPORARY VARIABLES FOR STORING THE MSB OF EACH 6-BIT INPUT FOR USE IN 2'S COMPLEMENT COMPARISON, ALONG WITH THE 1-BIT COMPARATOR RESULT
    wire msbA, msbB, out;
    assign msbA = i0[5];
    assign msbB = i1[5];

    //FOR EACH 8-BIT INPUT FOR THE >= CIRCUIT, LET BITS 0->5 = THE 6-BIT INPUT TO THE ALU AND LET THE OTHER BITS =0
    assign A8[7:6] = 0;
    assign A8[5:0] = i0;
    assign B8[7:6] = 0;
    assign B8[5:0] = i1;

    //PASS THESE CREATED 8-BIT NUMBERS TO THE 8-BIT >= CIRCUIT
    eq8 A_GREQ_B (.c(A8), .d(B8), .greq(out));

    //USING THE MSB OF EACH INPUT TO THE ALU AND THE RESULT OF THE >= CIRCUIT, THIS NOW WORKS AS A 2'S COMPLEMENT COMPARATOR
    assign two_comp_greq = ~msbA & out | ~msbA & msbB | msbB & out;

endmodule
```

Fig 5.4 – geq6 – using eq8 i.e. 8-bit comparator to check A >= B operation for 2 inputs

The following images constitutes the constraints for the Basys3 Board implementation. The implementation is further explained the demo 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 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 PACKAGE_PIN U16 [get_ports {x[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {x[0]}]
set_property PACKAGE_PIN E19 [get_ports {x[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {x[1]}]
set_property PACKAGE_PIN U19 [get_ports {x[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {x[2]}]
set_property PACKAGE_PIN V19 [get_ports {x[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {x[3]}]
set_property PACKAGE_PIN W18 [get_ports {x[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {x[4]}]
set_property PACKAGE_PIN U15 [get_ports {x[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {x[5]}]
```

Fig 5.4 – Constraints file implementation for Basys3 board

6. Demonstration (DEMO):

Before demonstration I would like to mention that I have implemented this using previous Lab components which I have attached including the modifications applied to them. My previous submissions are present on Blackboard.

Configuration Device: Basys3 (xc7a35tcpg236-1)
Target Language: Verilog

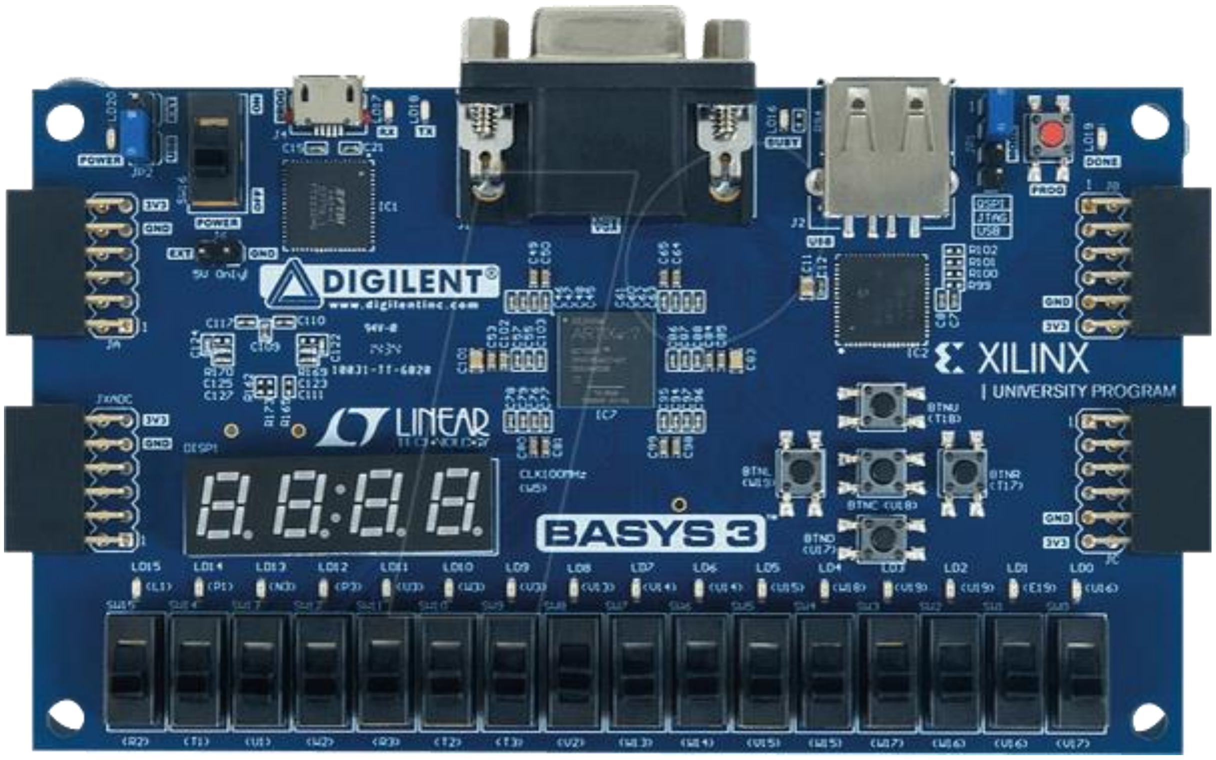


Fig 6.0 – Sample Basys3 Board

As mentioned in the constraint's implementation is defined as follows:

SWITCHES - *INPUT*			
V17 (LSB)	*A[0]*	**W14 (LSB)**	*B[0]*
V16	*A[1]*	**W13**	*B[1]*
W16	*A[2]*	**V2**	*B[2]*
W17	*A[3]*	**T3**	*B[3]*
W15	*A[4]*	**T2**	*B[4]*
V15 (MSB)	*A[5]*	**R3 (MSB)**	*B[5]*

V1	*FXN[0]*	**T1**	*FXN[1]*	**R2**	*FXN[2]*

The input is taken using the switches assigned. OFF switch returns 0 bit whereas ON switch returns 1 bit.

LEDs - *OUTPUT*	
U16 (LSB)	*X[0]*
E19	*X[1]*
U19	*X[2]*
V19	*X[3]*
W18	*X[4]*
U15 (MSB)	*X[5]*

The output is generated using the LEDs assigned. OFF led returns 0 bit whereas ON led returns 1 bit.

The expansion of output is slightly different for X[0] assigned to AGREQ which returned when A>=B fxn is triggered. The output lights up X[0] when the condition is satisfied. Shown in the test case.

Basys3 Board – Test Cases:

Case 0: **fxn = b'000**

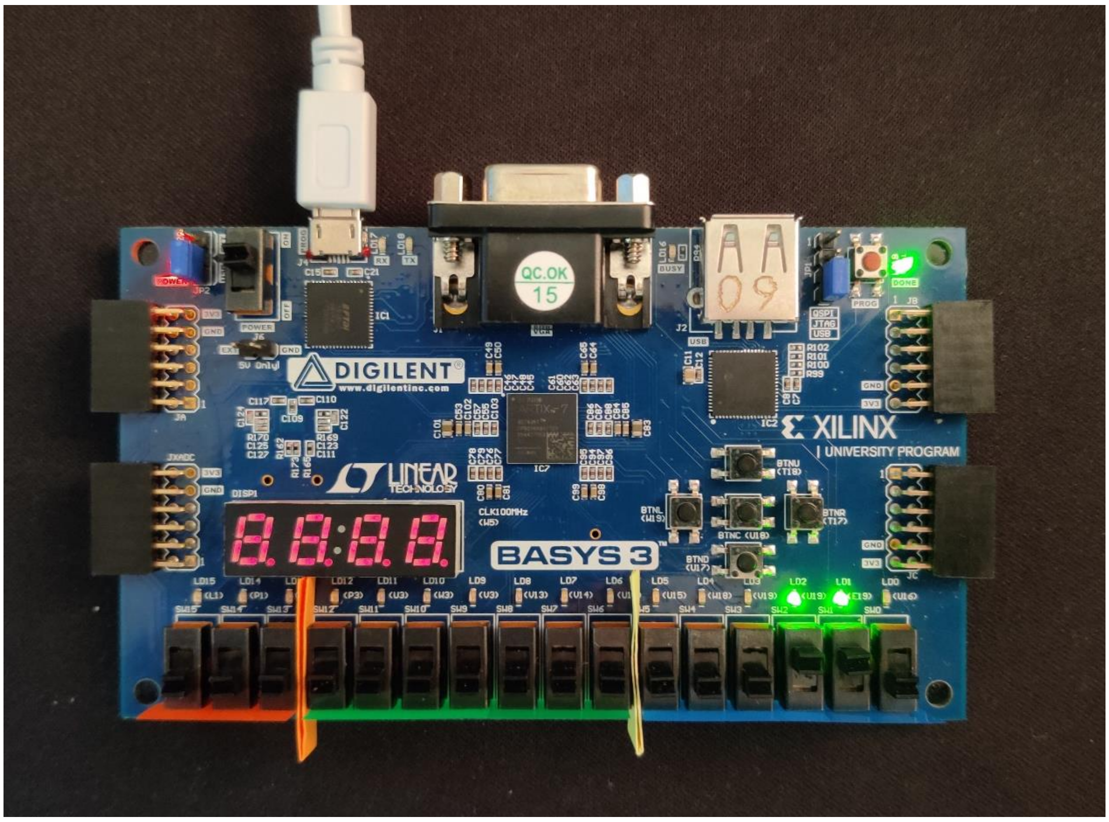


Fig 6.1 – display input A (000110) on board

Case 1: **fxn = b’001**

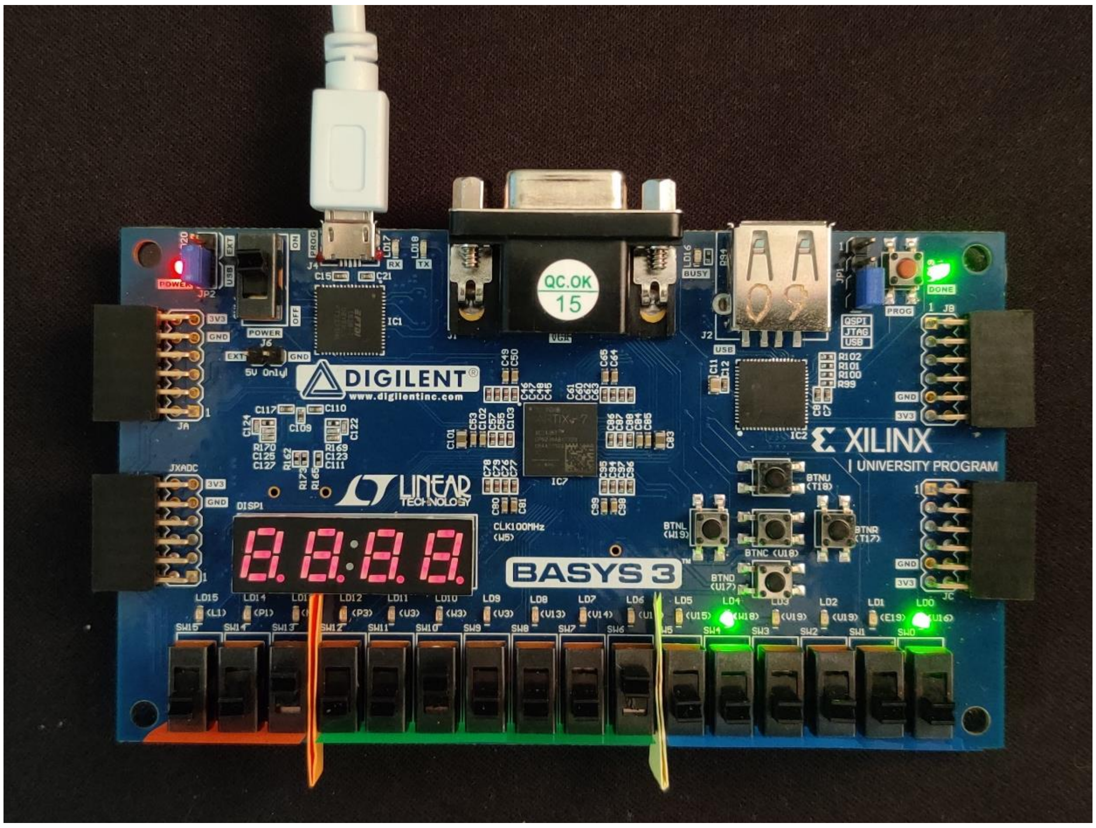


Fig 6.2 – display input B (010001) on board

Case 2: **fxn = b’010**

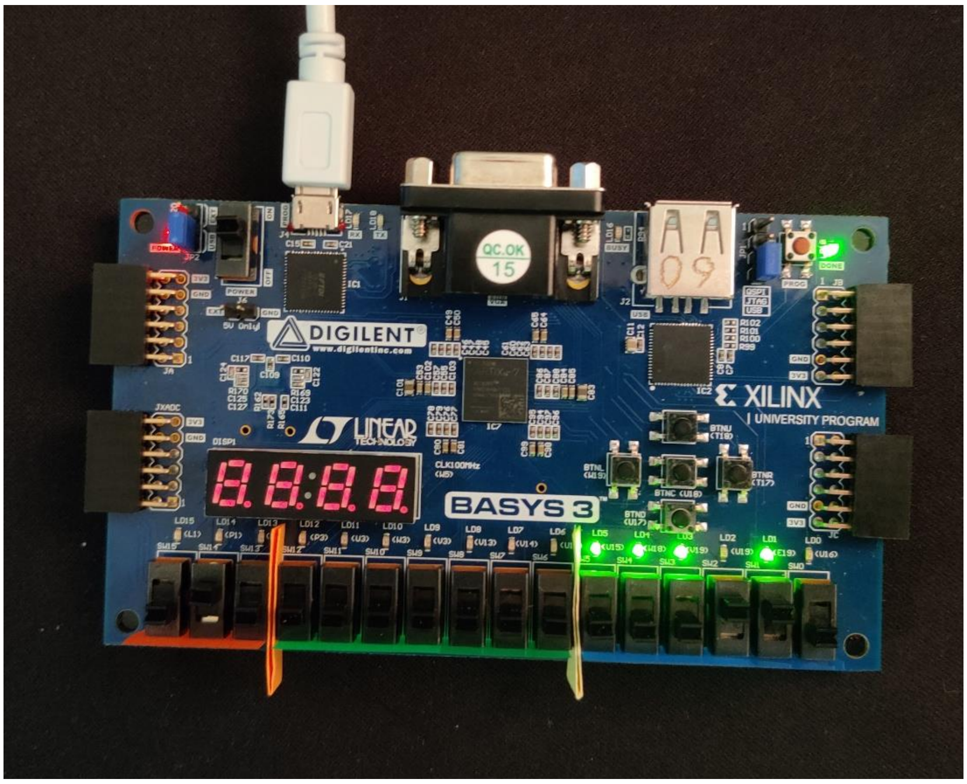


Fig 6.3 – display -A (2’s compliment of 000110 = 111010)

Case 3: **fxn = b'011**

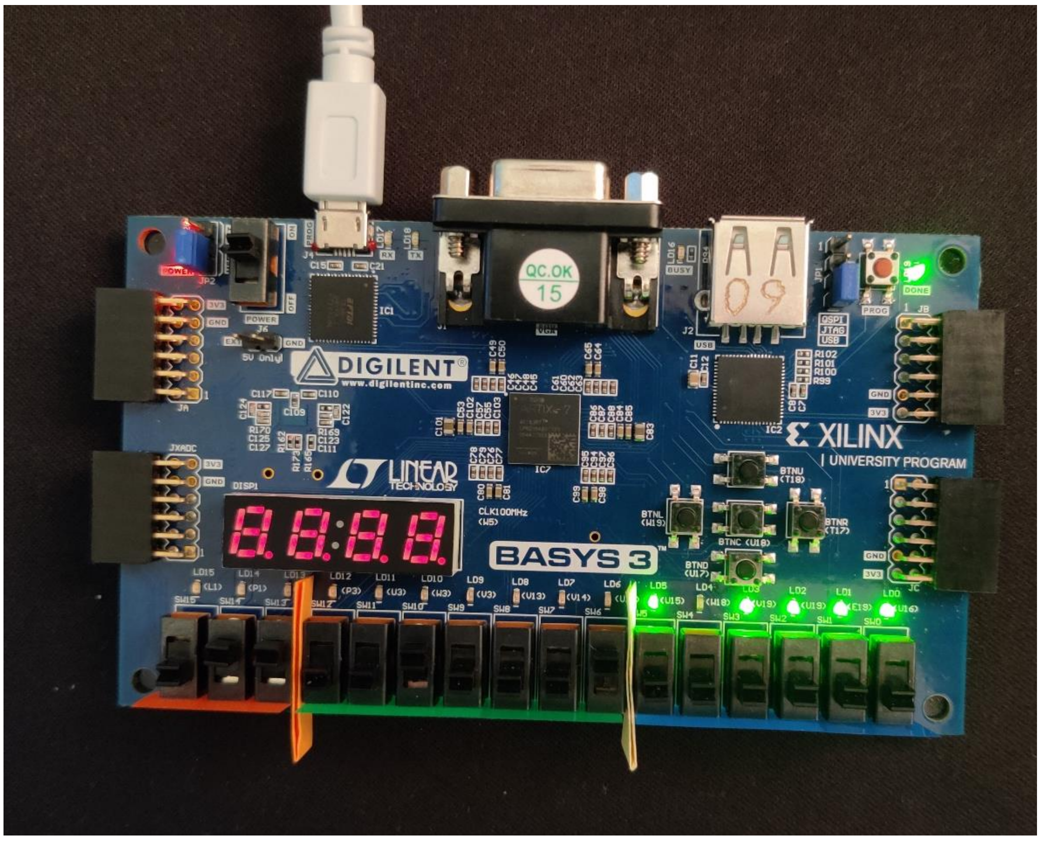


Fig 6.4 – display -B (2's compliment of 010001 = 101111)

Case 4: **fxn = b'100**

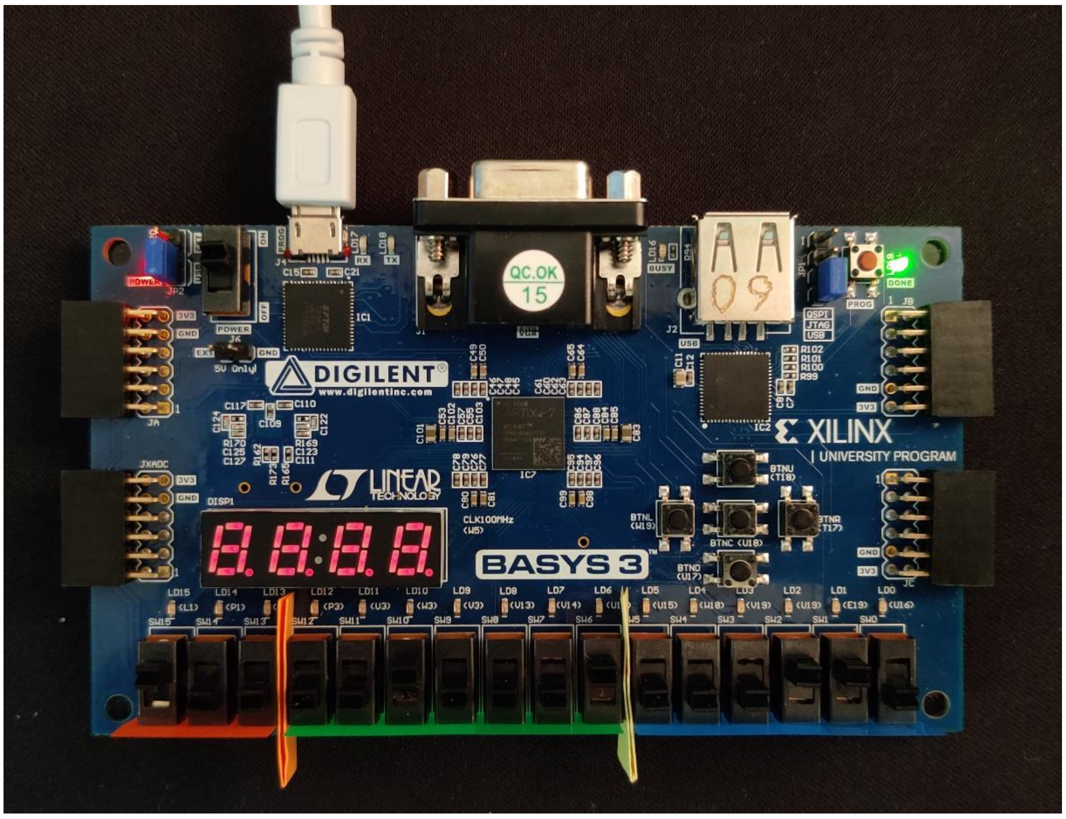


Fig 6.5.1 – display A (000110) >=B (010001) i.e. false – U16 doesn't light up

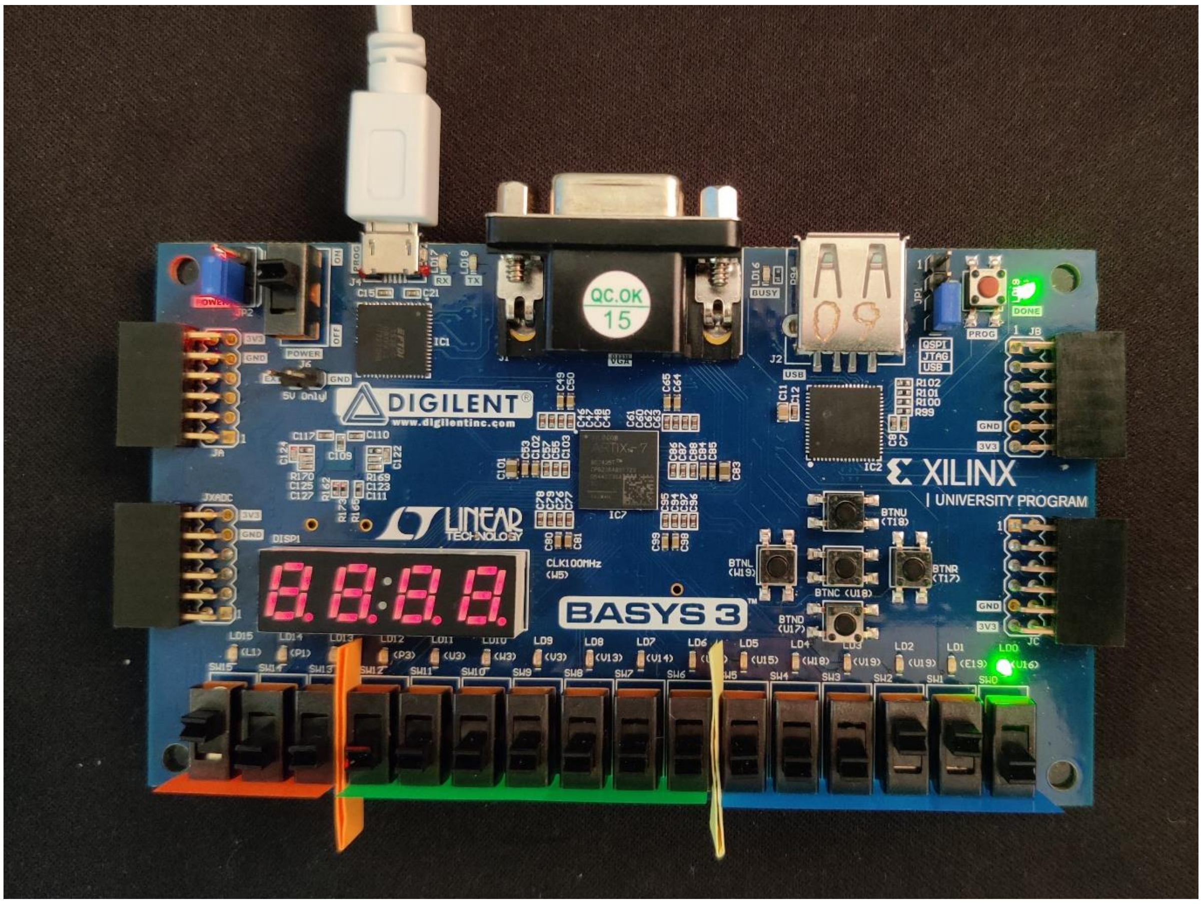


Fig 6.5.2 – display A (000000) >=B (000110) i.e. true – U16 lights up

Case 5: **fxn = b'101**

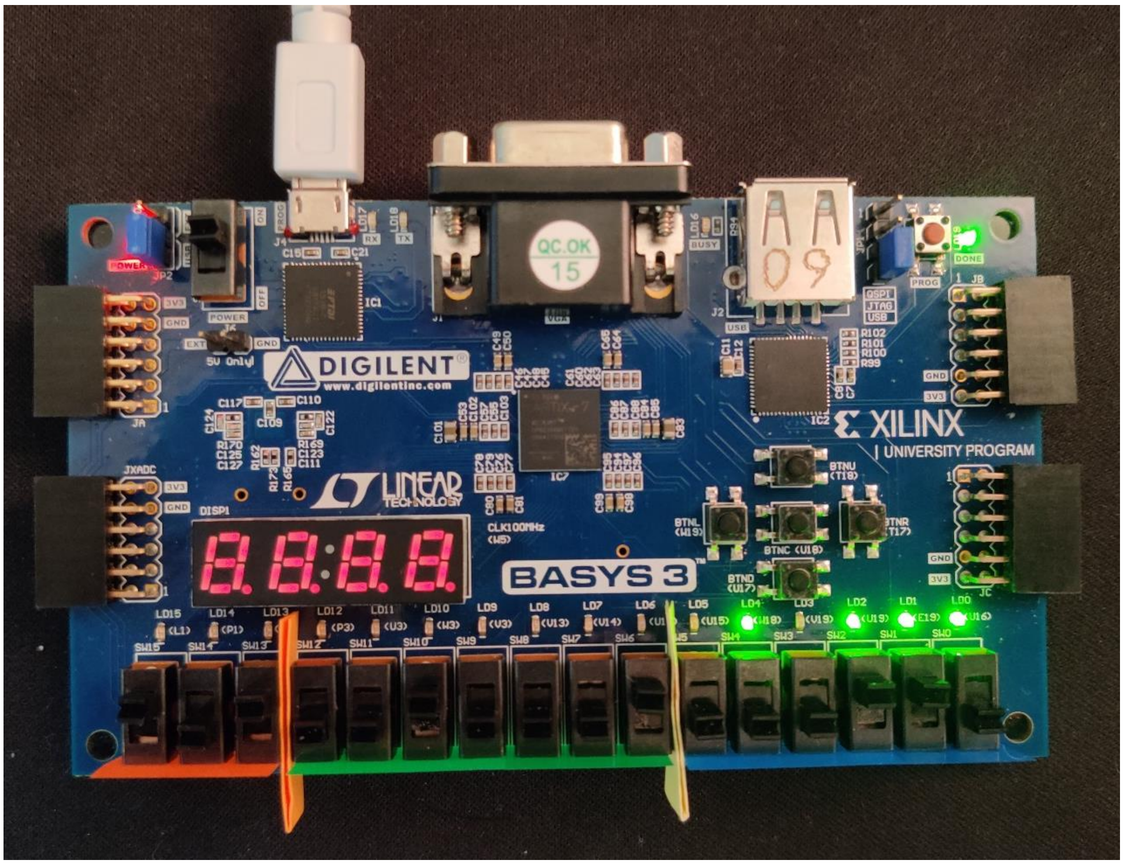


Fig 6.6 – display A (010001) ^ B (000110) = (010111)

Case 6: **fxn = b'110**

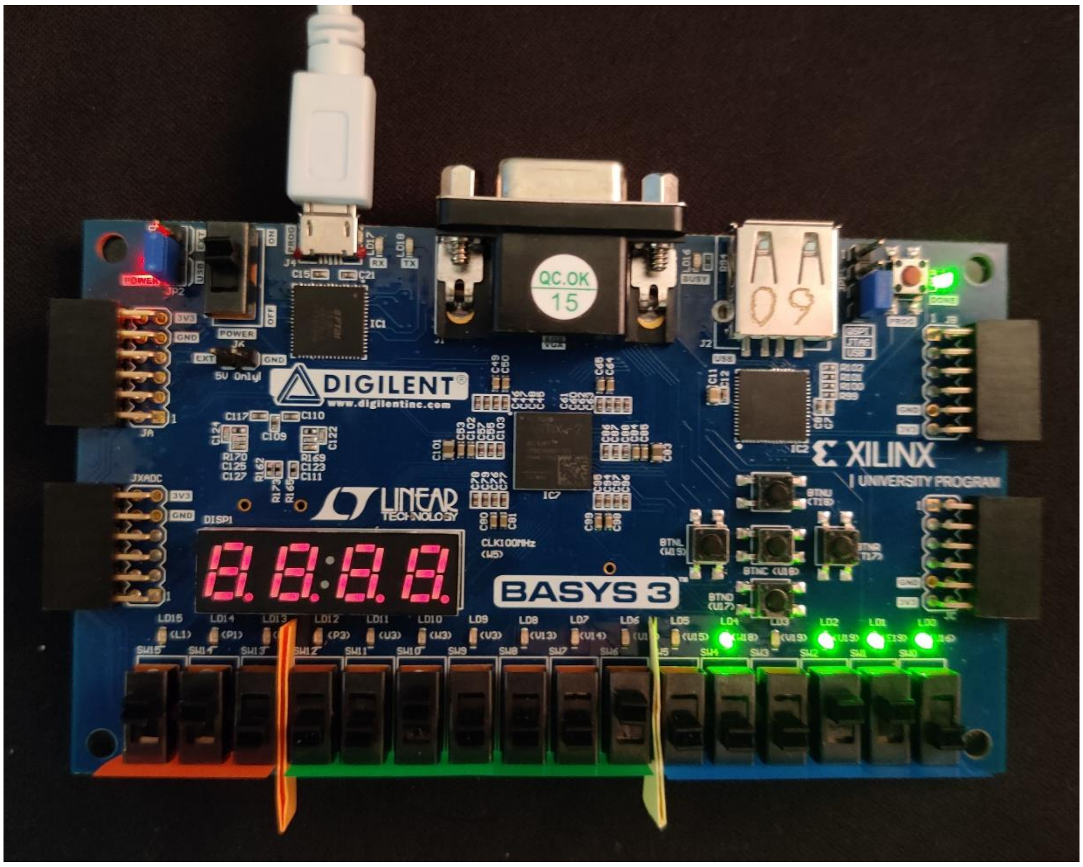


Fig 6.7 – display A (010001) + B (000110) = 010111

Case 7: **fxn = b'111**

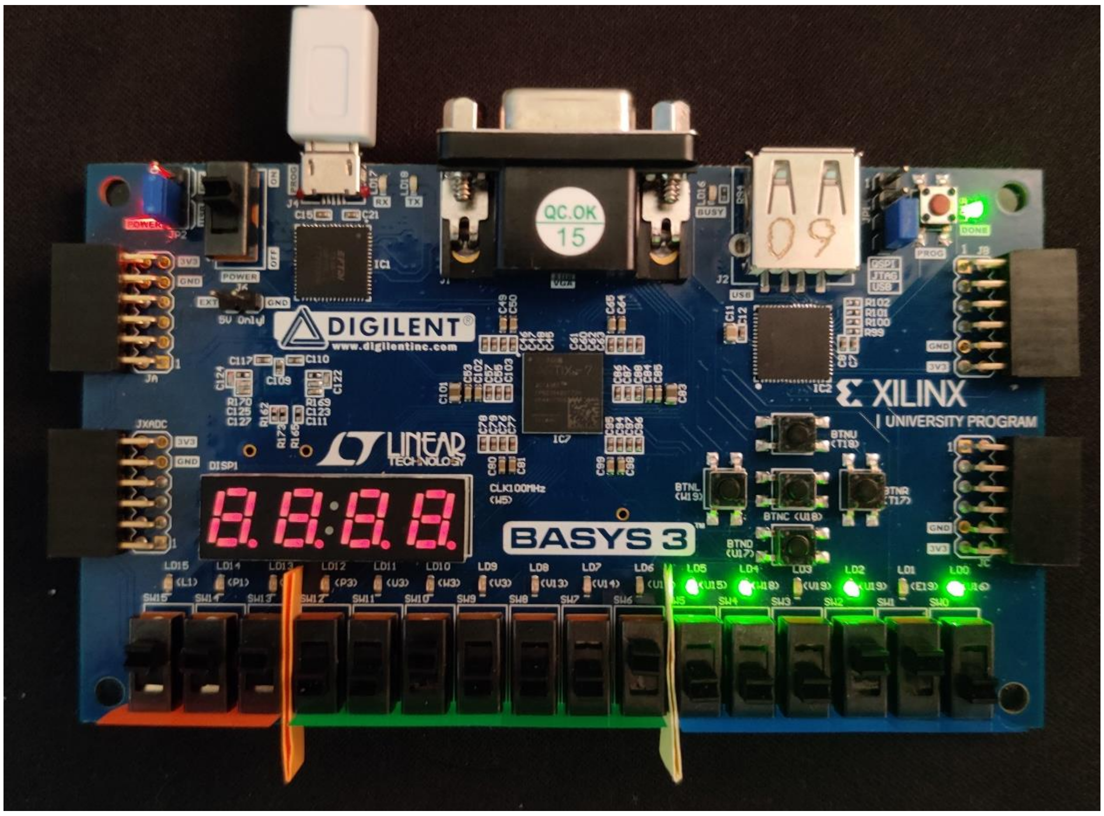


Fig 6.8 – display A (010001) - B (000110) = 110101

Conclusion:

Given the test-cases and functionality checks made on the testbench as well as the basys3 board. The implementation of the project with exhaustive testing of every module the mini ALU works well (even tested on edge cases). Each module used is declared. The modules uploaded in previous lab sessions are bit modified and attached to the assignment. I learned a lot more about system designing and implementation using Verilog.

Appendix

Inclusive to LabB_tanejar, eq8, geq2, eq2, eq1, 6bit_ripple_adder, full_adder, and LabC_tanejar which were presented on time, this submission concludes with attached modules as stated in Fig. 3.3 and mini_alu.bit for testing on the board.