3C7 Digital System Design

Assignment 1

Shane O'Donnell – 21364002

Lab Description:

You need to design, write/modify the Verilog modules for the following functions, and test a "mini" arithmetic logic unit. An arithmetic logic unit (ALU) uses combinatorial logic to implement common arithmetic and logical 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 less than B)
101	(A *nxor* B) (Bitwise XNOR)
110	A+B
111	A-B

Implementation:

The implementation of the arithmetic logic board involved initializing multiple modules from previous laboratories as well as editing previous labs to conform to 6 bit binary numbers in 2's complement form, and finally, some modules were to be built from scratch (A nxor B , -A, -B). Modules such as A + B, A – B, and A<B were taken and edited from previous labs.

Simulation:

In order to ensure all modules were functioning correctly, a testbench was implemented in order to test each of the nine modules separately and eliminate any errors. Figure 1 displays the testcases implemented for the function call 000 (X = A). The value of B was kept constant at zero in order to ensure that the output was solely utilizing the value in the A input for this function call. The first test vector is set as the 6 least significant bits of my personal board number 69. Figure 2 displays the same test cases in binary notation.

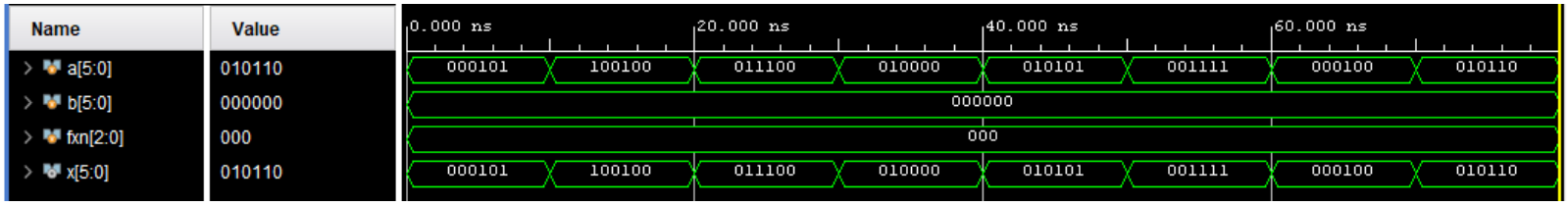

Figure 1: fxn = b'000 , output displays A

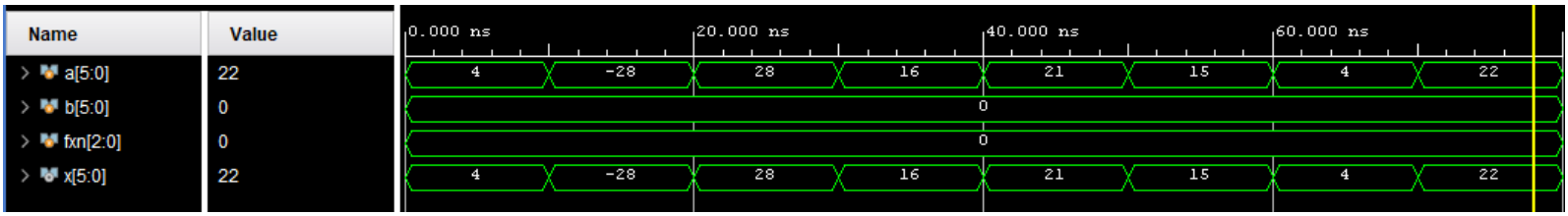

Figure 2: fxn = b'000 in signed decimal format

Figure 3 displays the function call 001 (X = B), the A input value was set constant as 0 to ensure that the output was solely to do with the B input.

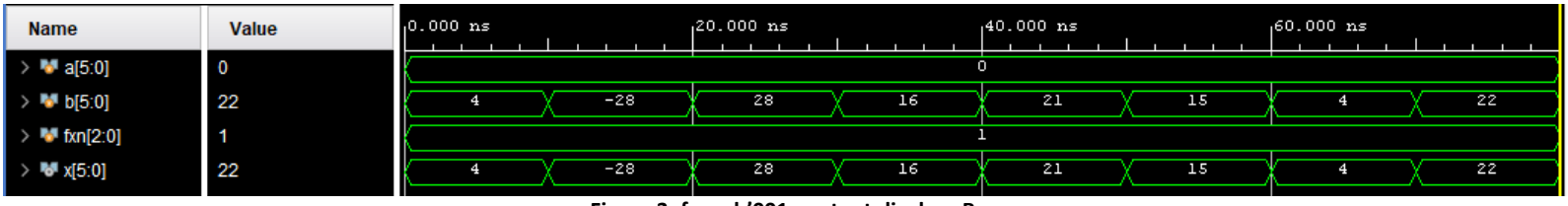

Figure 3: fxn = b'001 , output displays B

Figure 4 displays the function call 010 (X = -A) The value of B was kept constant at zero in order to ensure that the output was solely utilizing the value in the A input for this function call.

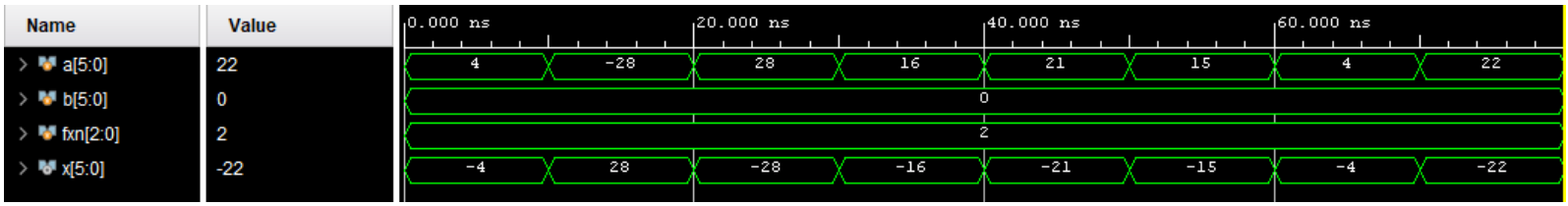

Figure 4: fxn = b'010 , output displays -A

Figure 5 displays the function call 011 (X = -B) The value of A was kept constant at zero in order to ensure that the output was solely utilizing the value in the B input for this function call.

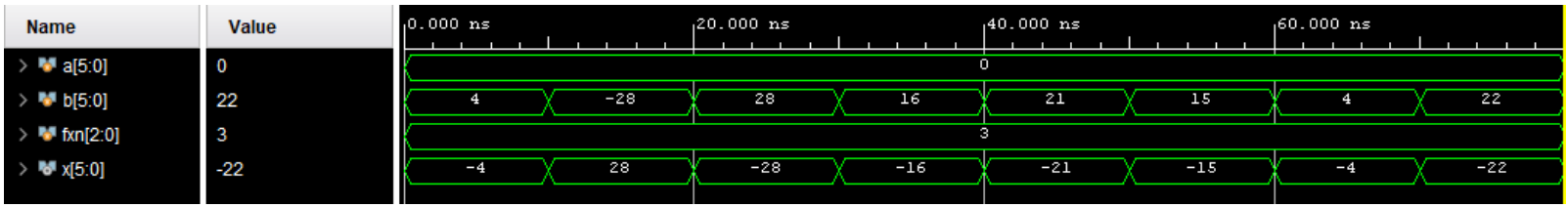

Figure 5: fxn = 011 , output displays -B

Figure 6 displays the output for the function call 100 (X = (A<B)).

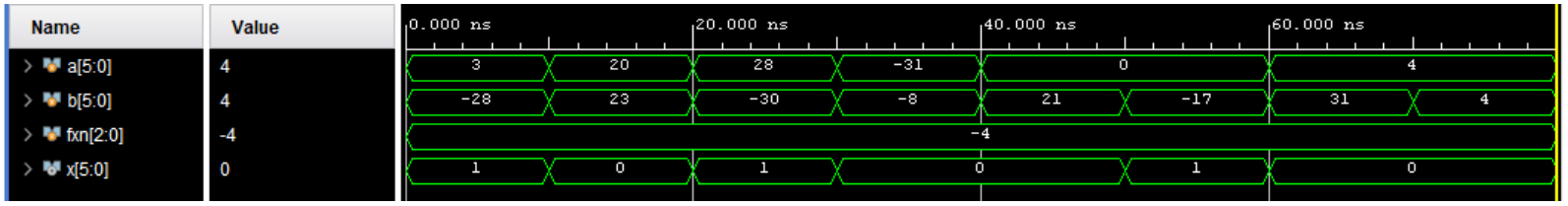

Figure 6: fxn = b'100 , output displays A<B

Figure 7 displays the test bench for the function call 101 (X = A nxor B , Bitwise nxor)

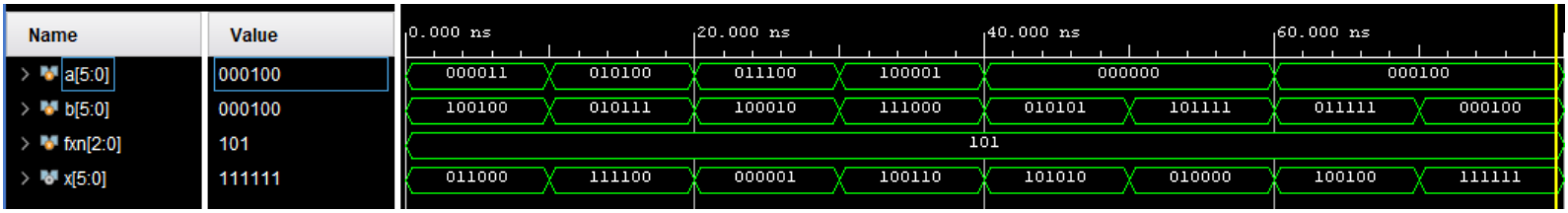

Figure 7: fxn = b'101 , output displays A nxor B (BITWISE NXOR)

Figure 8 displays the test bench for the function call 110 (X = A + B).

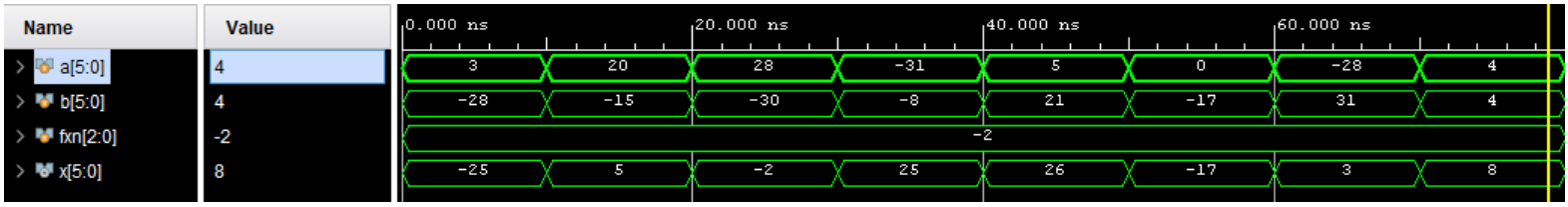

Figure 8: fxn = b'110 , output displays A + B

Figure 9 displays the test bench for the function call 111 (X = A + B).

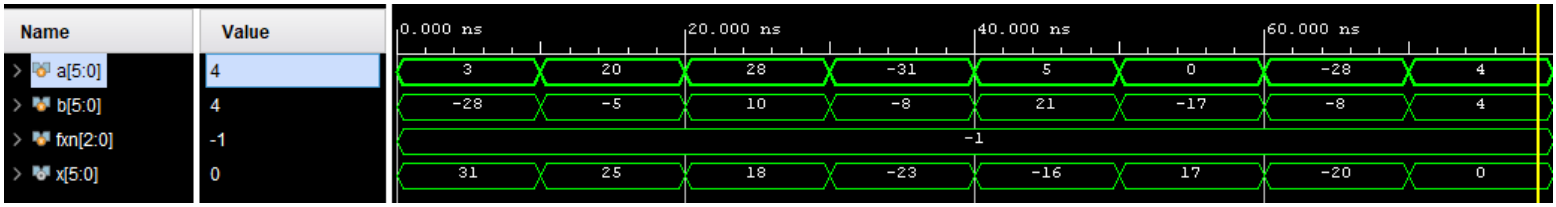

Figure 9: fxn = b'111 , output displays A - B

Vivado Environment Set up:

In order to set up the Arithmetic Logic Unit, submodules were initialized, in order to call these operation functions inside the ALU module. Figure 10 displays the hierarchy of the modules, inside each function submodule there are more submodules which ensure each function functions correctly. These files are displayed in figure 11.

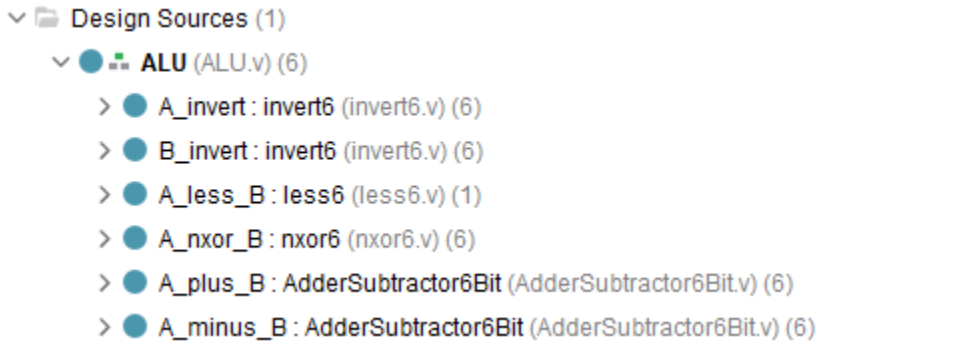

Figure 10: Verilog Hierarchy

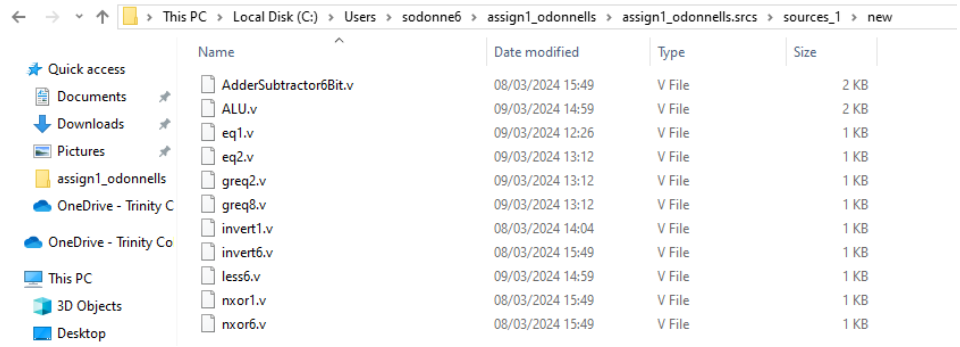

Figure 11: ALU Source Files

Schematic:

In order to comprehend the ALU's functionality the following schematics were generated using Verilog. Figure 12 displays the register transfer level schematic generated by Verilog. As seen in the schematic, the ALU has three major input values which are, a, b, and fxn. A and b store the two 6 bit numbers used for the arithmetic while the fxn input stores the function number call which decides what arithmetic function will be used.

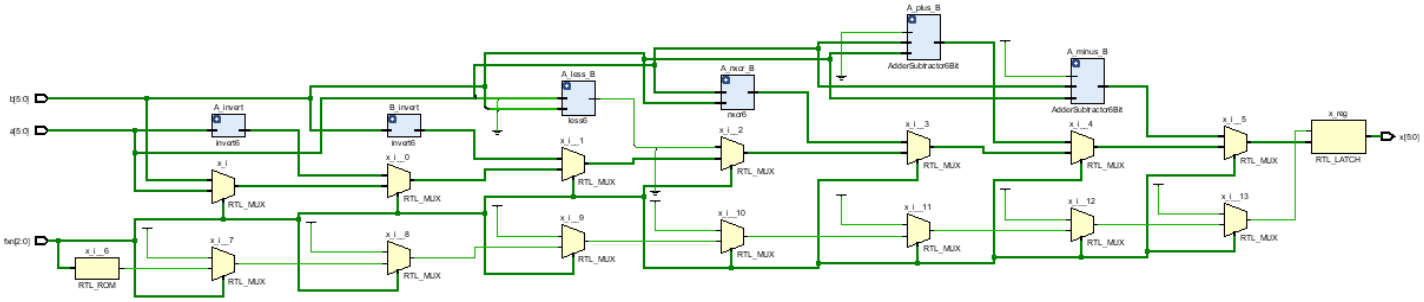

Figure 12: Register Transfer Level Schematic

As well as this, figure 13 displays the synthesis schematic for the ALU.

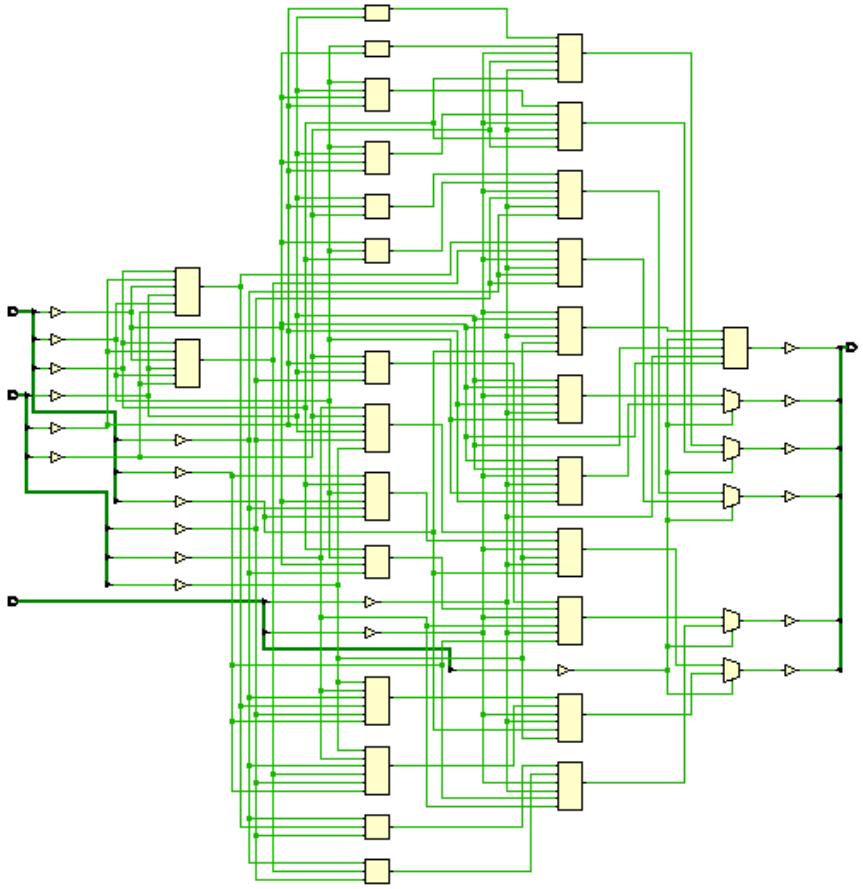

Figure 13: Synthesis Schematic

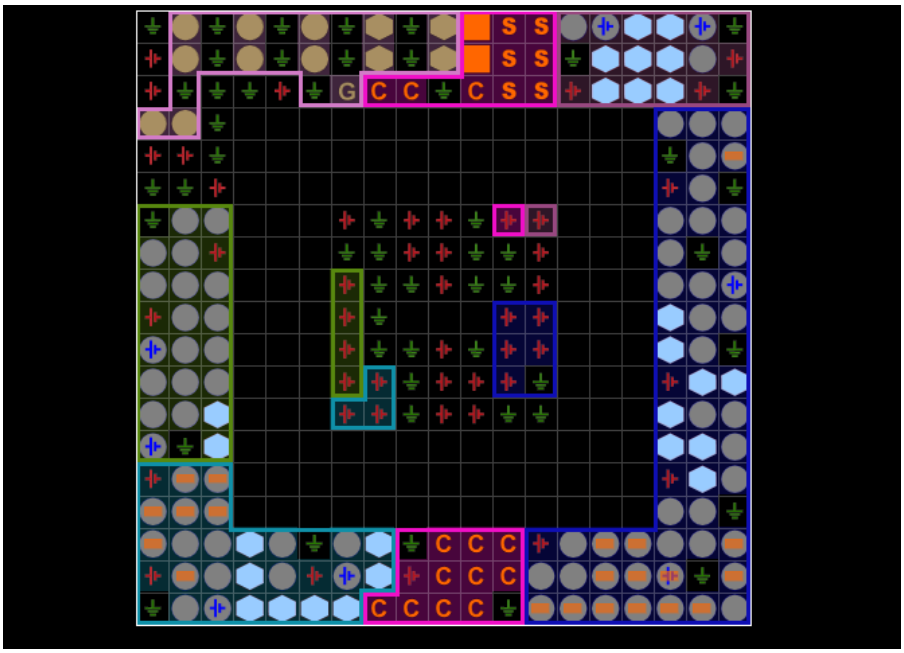

Figure 14: Implemented Package

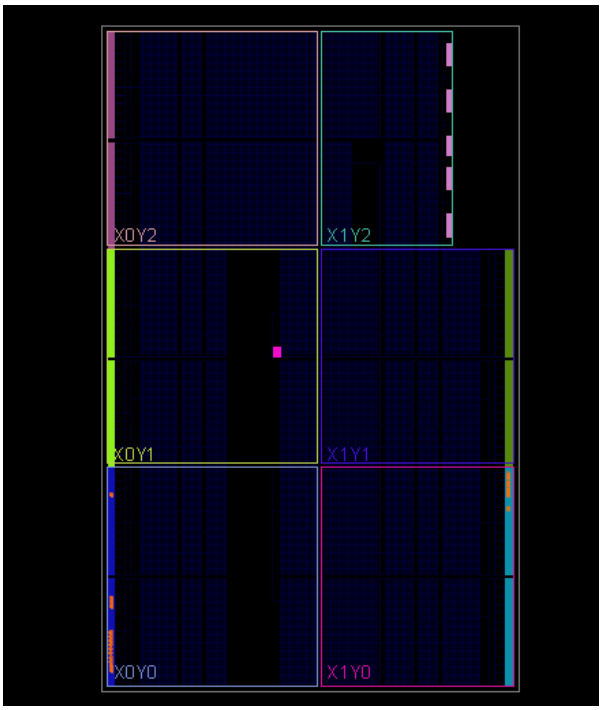

Figure 15: Implemented Device

ALU Module Code:

In order to implement the ability to call each function separately, the following module was created to control the ALU. It can be seen that on start up, each function is called and executed first, after this eight if statements are used to match the function number with each function. For the A is less than B function, only the least significant bit is outputted as the final output is either true or false, all other functions are outputted as a full 6 bit binary number.

```
module ALU(
input wire [5:0] a,b,
input wire [2:0] fxn,
output reg [5:0] x
    );

wire [5:0] Ainvert,Binvert,AnxorB,AplusB,AminusB;          //6 bit output
wire AlessB;                                               //Output is 1 bit so initialise as 1bit and alter in subroutine

invert6 A_invert (.x(a), .invertx(Ainvert));                                  //X -> -A
invert6 B_invert (.x(b), .invertx(Binvert));                                  //X -> -B
less6 A_less_B (.x(a), .y(b), .less_6(AlessB));                               //X -> A<B
nxor6 A_nxor_B (.x(a),.y(b), .nxor_6(AnxorB));                                //X -> A nxor B (BITWISE NXOR)
AdderSubtractor6Bit A_plus_B (.x(a), .y(b), .sel(0), .sum(AplusB));           //X -> A + B
AdderSubtractor6Bit A_minus_B (.x(a), .y(b), .sel(1), .sum(AminusB));         //X -> A - B

//sort function calls
always @(*)
begin
    if (fxn == 3'b000) {x} = {a};                                  // X -> A                        //DONE
    if (fxn == 3'b001) {x} = {b};                                  // X -> B                        //DONE
    if (fxn == 3'b010) {x} = {Ainvert};                            // X -> -A                       //DONE
    if (fxn == 3'b011) {x} = {Binvert};                            // X -> -B                       //DONE
    if (fxn == 3'b100) {x[5:1],x[0]} = {0,AlessB};                 // X -> A<B
    if (fxn == 3'b101) {x} = {AnxorB};                             // X -> A nxor B (BITWISE NXOR)  //DONE
    if (fxn == 3'b110) {x} = {AplusB};                             // X -> A + B                    //DONE
    if (fxn == 3'b111) {x} = {AminusB};                            // X -> A - B                    //DONE

end
endmodule
```

Figure 16: ALU Module

Demo

In order to implement the ALU onto the BASYS 3 board, the constraint file displayed in figure 17 was used. To ensure the inputs and output values could be clearly displayed on the board, the six most right switches were initialized as the a inputs, the next 6 most right switches were initialized as the b inputs, then the three most left switches were initialized as the function call inputs. In order to display the output value the 6 most right LED's were initialized as the x output bits. The input and

```
10
11  ## Switches
12  #A switches
13  set_property PACKAGE_PIN V17 [get_ports {a[0]}]
14      set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
15  set_property PACKAGE_PIN V16 [get_ports {a[1]}]
16      set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
17  set_property PACKAGE_PIN W16 [get_ports {a[2]}]
18      set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]
19  set_property PACKAGE_PIN W17 [get_ports {a[3]}]
20      set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]
21  set_property PACKAGE_PIN W15 [get_ports {a[4]}]
22      set_property IOSTANDARD LVCMOS33 [get_ports {a[4]}]
23  set_property PACKAGE_PIN V15 [get_ports {a[5]}]
24      set_property IOSTANDARD LVCMOS33 [get_ports {a[5]}]
25  #B switches
26  set_property PACKAGE_PIN W14 [get_ports {b[0]}]
27      set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
28  set_property PACKAGE_PIN W13 [get_ports {b[1]}]
29      set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
30  set_property PACKAGE_PIN V2 [get_ports {b[2]}]
31      set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
32  set_property PACKAGE_PIN T3 [get_ports {b[3]}]
33      set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
34  set_property PACKAGE_PIN T2 [get_ports {b[4]}]
35      set_property IOSTANDARD LVCMOS33 [get_ports {b[4]}]
36  set_property PACKAGE_PIN R3 [get_ports {b[5]}]
37      set_property IOSTANDARD LVCMOS33 [get_ports {b[5]}]

48
49  ##LEDs
50  set_property PACKAGE_PIN U16 [get_ports {x[0]}]
51      set_property IOSTANDARD LVCMOS33 [get_ports {x[0]}]
52  set_property PACKAGE_PIN E19 [get_ports {x[1]}]
53      set_property IOSTANDARD LVCMOS33 [get_ports {x[1]}]
54  set_property PACKAGE_PIN U19 [get_ports {x[2]}]
55      set_property IOSTANDARD LVCMOS33 [get_ports {x[2]}]
56  set_property PACKAGE_PIN V19 [get_ports {x[3]}]
57      set_property IOSTANDARD LVCMOS33 [get_ports {x[3]}]
58  set_property PACKAGE_PIN W18 [get_ports {x[4]}]
59      set_property IOSTANDARD LVCMOS33 [get_ports {x[4]}]
60  set_property PACKAGE_PIN U15 [get_ports {x[5]}]
61      set_property IOSTANDARD LVCMOS33 [get_ports {x[5]}]
```

Figure 17: .xdc file – Switch and LED initialization

The following tables display how the input and output pins have been assigned to the basys 3 board.

Input Switches - a and b			
V17	a[0]	W14	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	a[5]	R3	b[5]

Figure 18: Input switches for A and B

Input Switches - fxn					
V1	fxn[0]	T1	fxn[1]	R2	fxn[2]

Figure 19: Input switches for function numbers

Output LED's	
U16	x[0]
E19	x[1]
U19	x[2]
V19	x[3]
W18	x[4]
U15	x[5]

Figure 20: Output LED's for X

After ensuring all the function cases were functioning correctly and were implemented correctly into the ALU module it was possible to load and execute the ALU's functionality onto the BASYS 3 board. In order to ensure the functions also functioned correctly on the board the following test cases were completed for each function call. In order to ensure the switch functionality is clear, paper dividers were utilized to represent the boards of each input set.

B’000

The first function call outputs the 6 bit binary number created using the a input switches. This can be seen in figure 21.

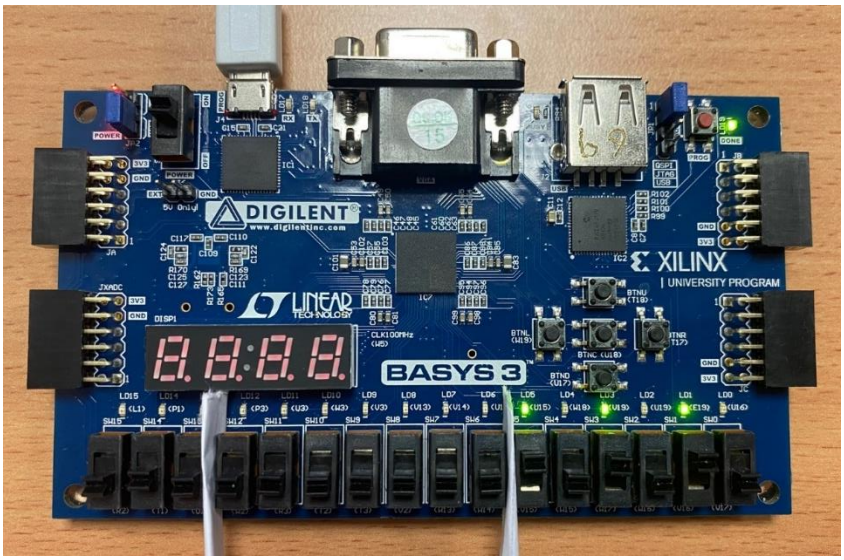

Figure 21: fxn = b’000 , output displays A

B’001

The second function call outputs the 6 bit binary number created using the b input switches. This can be seen in figure 22.

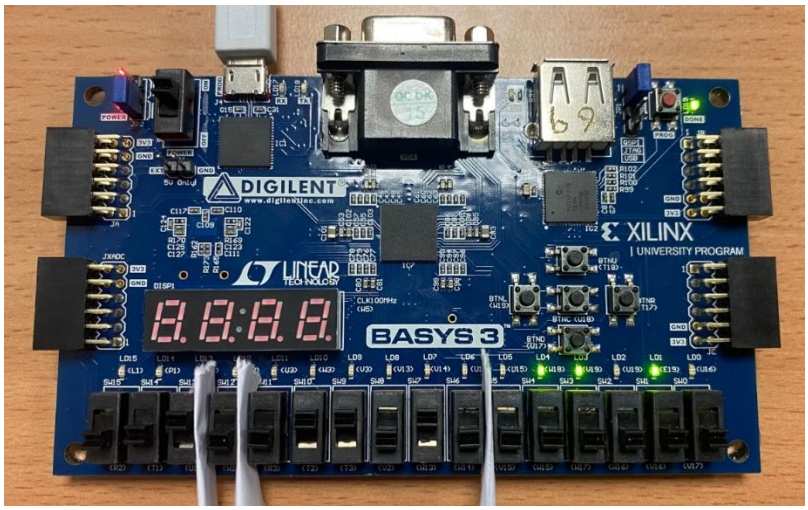

Figure 22: fxn = b’001, output displays B

B’010

The third function call inverts the 6 bit binary number created using the A input switches, figure 23 displays this inversion by inputting 4 (000100) which is inverted as displays a -4 (111100).

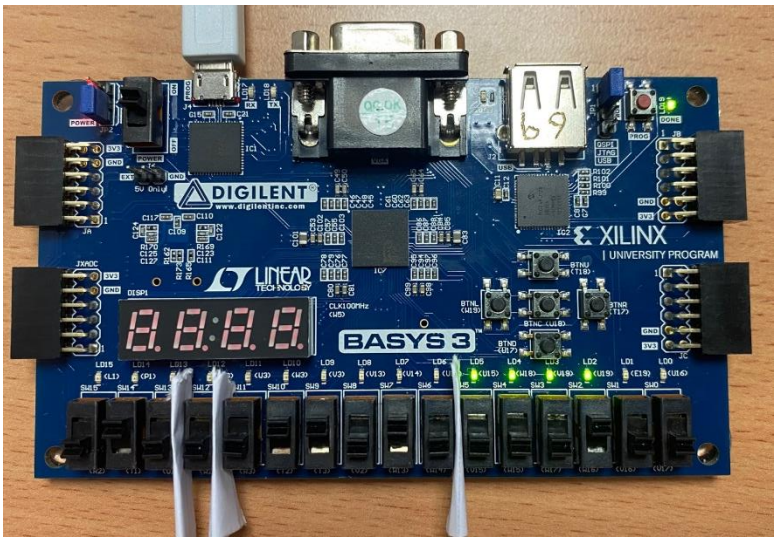

Figure 23: fxn = b’010 , output displays -A

B’011

The fourth function call inverts the 6 bit binary number created using the B input switches, figure 24 displays this inversion by inputting 11 (001011) which is inverted as displays a -11 (110101).

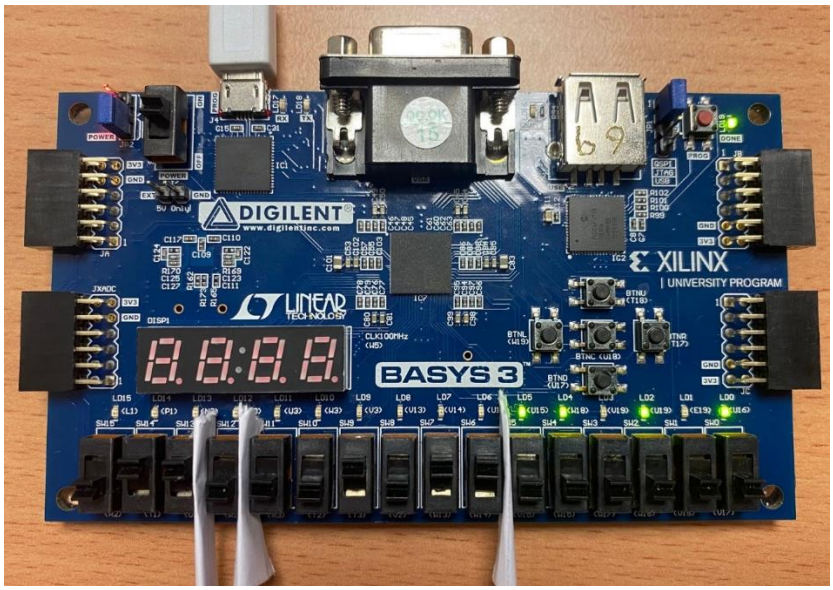

Figure 24: fxn = b’011 , output displays -B

B’100

The fifth function call checks if the 6 bit binary number created using the B switches is less than the 6 bit binary number created using the A switches. Figure 25 displays an A input of 1 (000001), and a B input of -31 (100001). Only 1 output led is needed to display the output for this function; due to B being less than A in this test case the LED is set on.

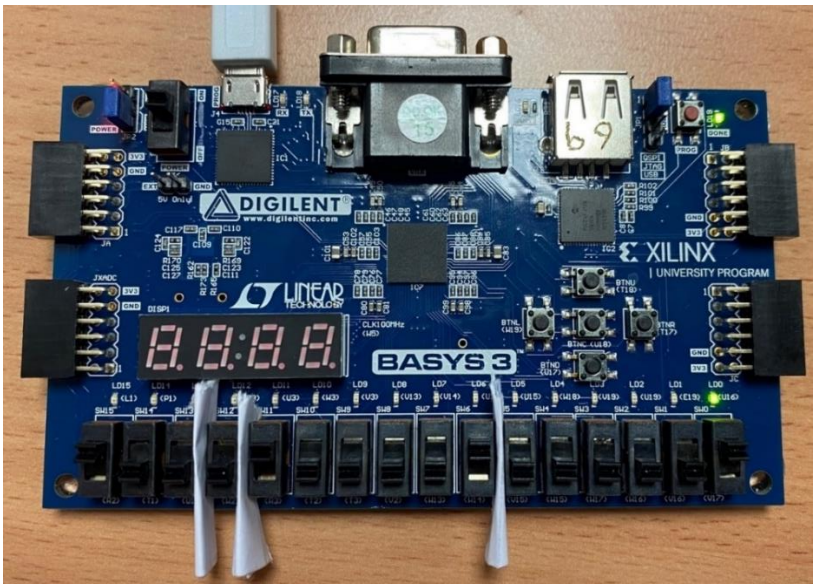

Figure 25: b’100 , output displays (A<B)

B'101

The sixth function call is a bitwise nxor for the two input values A and B. Figure 26 displays the logic table for an nxor gate, essentially the output is only 1 when both the input values are the same. As well as this, figure 27 displays an A input of 000100 and a B input of 010101 which leads to an output of 101110.

NXOR Logic Table		
A	B	out
0	0	1
1	0	0
0	1	0
1	1	1

Figure 26: NXOR Logic Table

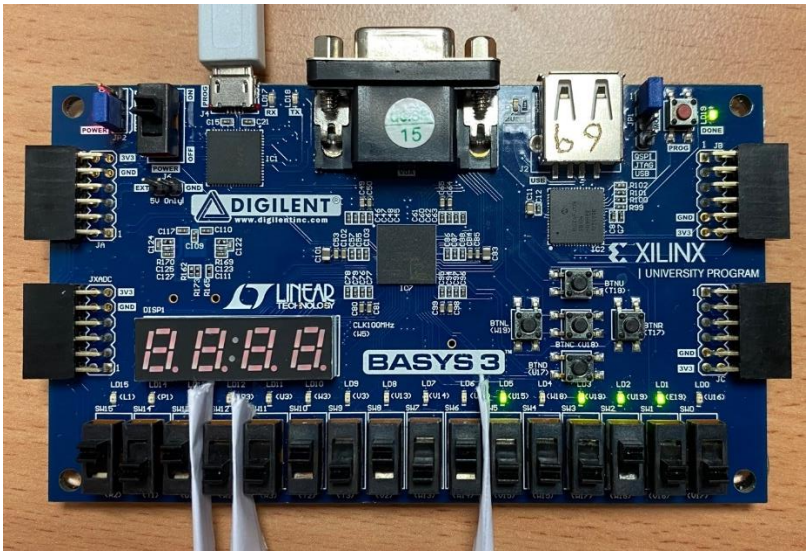

Figure 27: b'101 , output displays A nxor B

B’110

The seventh function call acts as an addition function for the A and B inputs. Figure 28 displays the addition of 3 (000011) and 5 (000101) which outputs an 8 (001000) using the LED’s.

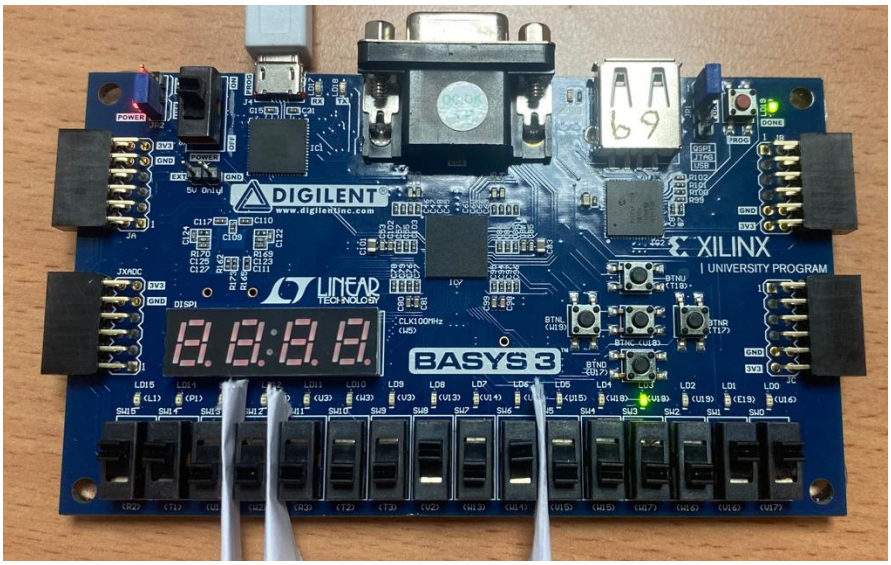

Figure 28: b’110 , output displays A + B

B’111

The eighth and final function call is a subtracter function which subtracts the B input from the A input. Figure 29 displays the input A which is 9 (001001) being subtracted by -3 (111101) which ouputs the value 12 (001100) using the LED’s.

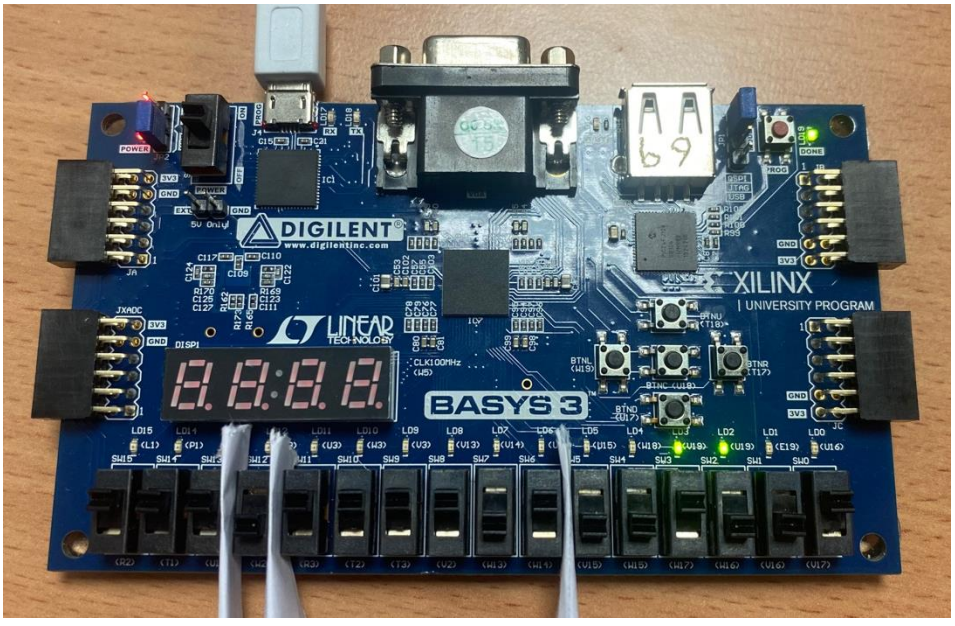

Figure 29: b’111 , output displays A - B

After implementing the ALU onto the Basys 3 board, a bit file was created corresponding to the modules loaded onto the board, figure 30 displays where this bit file was saved.

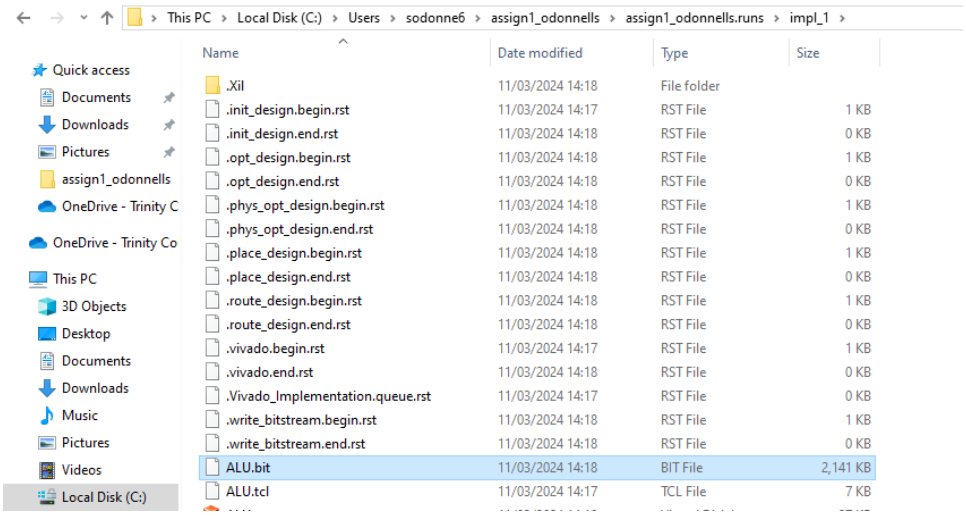

Figure 30: .bit file

Conclusion:

To conclude, the function calls specified at the beginning of this report were all successfully implemented; as well as this no errors or discrepancies were found while testing each function using a test bench or while testing the functions on the basys 3 board. Although, due to the inputs being in 6 bit 2's complement form, if a number greater than 31 or less than – 32 is inputted or outputted the arithmetic will be incorrect as the number will be out of the range of possibility implemented in the program.

Appendices:

All code which has been taken and edited from previous submissions can be found in the zip file provided in the submission.

LAB B

- eq1
- eq2
- greq2
- greq8

LAB C

- fulladder.v