Assignment 1 - Report

Stephen Komolafe (21336975)

In order to successfully implement the Arithmetic Logic Unit (ALU), several modules from previous labs had to be used. These modules include the 6bit_ripple_adder.v made in lab C, along with full_adder.v, eq1.v, and eq2.v sourced from Blackboard. Additionally, new modules such as FXN.v, ALU.v, ALU_Testbench.v, a 2's complement module for the ripple adder (SB_2sComp.v), and arithmetic function modules (AltB.v, AxnorB.v, etc) were created to ensure optimal ALU functionality. During the ALU testing phase, individual test vectors were formulated for each arithmetic operation, with 6-bit binary values assigned to inputs A and B. Given that a test vector was made for every arithmetic operation, a total of eight test vectors are present, each represented by a 3-bit binary value ranging from 000 to 111. The output for each operation is displayed as a 6-bit binary X value. The graphical representation of these operations can be observed in the waveform below.

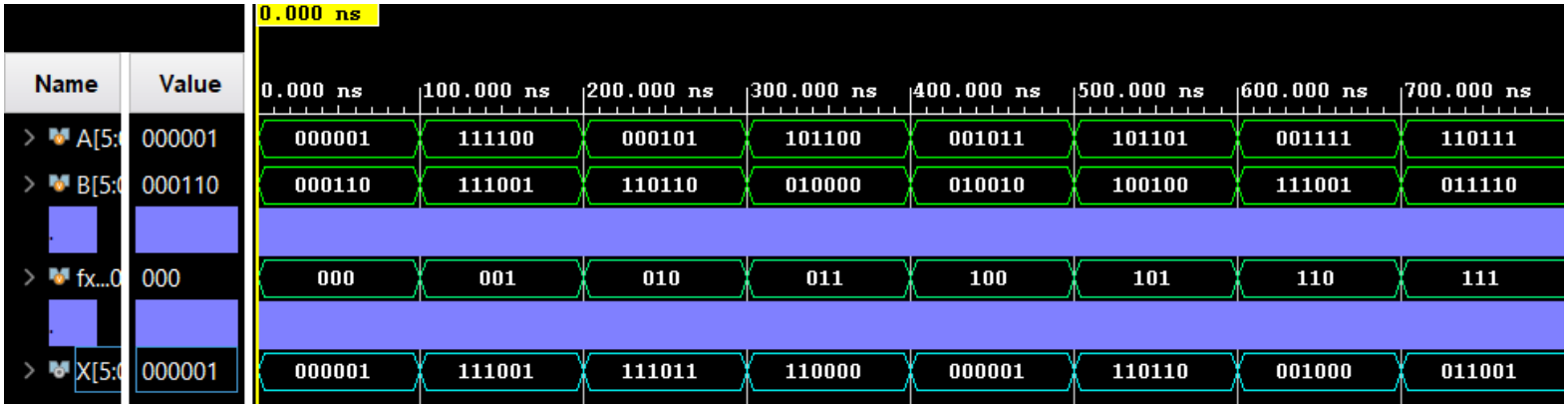


Figure 1. ALU Testbench Waveform in Binary

Each test vector performed as anticipated, yielding reasonable outputs for each arithmetic operation. To improve my testing approach, I would consider refining my test strategy for future labs by conducting multiple tests for each operation. However, considering the current compactness of the waveform, with only one test per operation, doubling or tripling the number of tests may introduce challenges in interpreting the waveform.

Synthesis and Implementation

The switch and LED pins in the constraints file Basys3_Master.xdc, provided through Blackboard, underwent adjustments to bring it in line with the inputs and outputs specified in the design. Following this, the design was synthesized and implemented in order to generate the bitstream necessary for programming the Basys-3 Board. Once programmed the design can be demonstrated on the board.

Demo

(Note: this is w.r.t pin values being read from right to left [i.e. (V17 –> V15) = V17, V16, W16, W17, W15, V15])
The six rightmost switches on the board (V17 –> V15) control the value of each bit in the 6-bit binary A input. The six bits to the left of V15 (W14 –> R3) control the value of each bit in the 6-bit binary B input and the three leftmost switches on the board control each bit of the 3-bit binary fxn input in order to indicate which arithmetic operation to use (A+B, A-B ,A XNOR B, etc.). That leaves the W2 switch, which serves no purpose. The six rightmost LEDs on the board (LD0 –> LD5) are used to depict the value of X outputted.

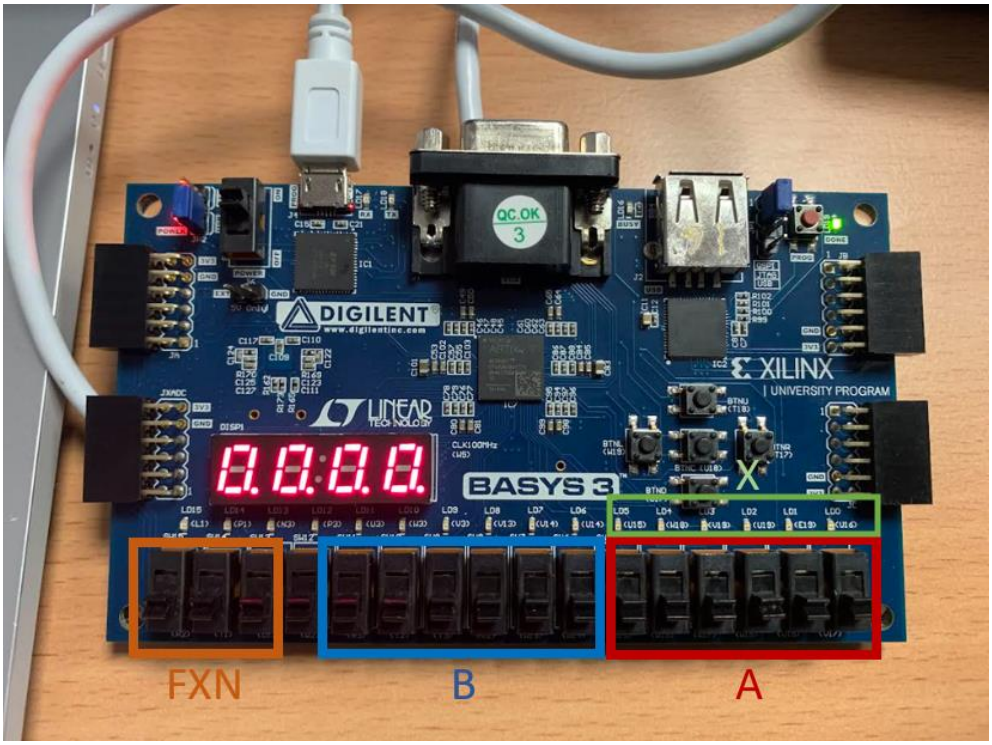


Figure 2. Basys-3 Board showing specific switches and LEDs used to control/Display ALU input and output

While controlling the board with the demonstrative guidelines outlined in the preceding paragraph in mind, the examination of the test vectors proceeded as follows:

Test Vectors

Test Vector 1. (A)

A	000001
B	000110
fxn	000
X	000001

Test Vector 2. (B)

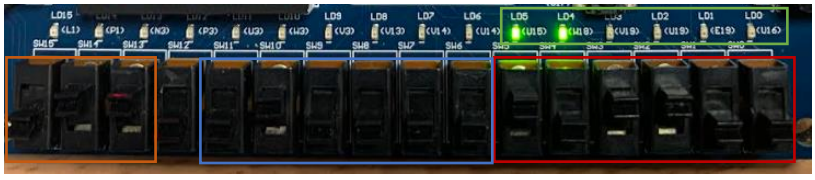
A	111100
B	111001
fxn	001
X	111001

Test Vector 3. (-A)

A	000101
B	110110
fxn	010
X	111011

Test Vector 4. (-B)

A	101100
B	010000
fxn	011
X	110000

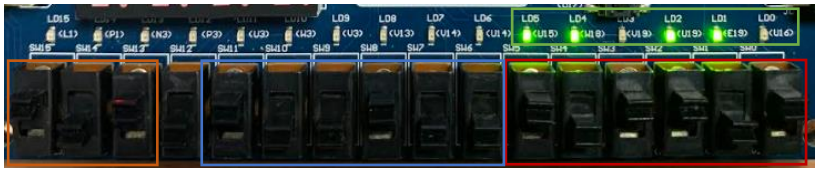


Test Vector 5. (A<B)

A	001011
B	010010
fxn	100
X	000001

Test Vector 6. (AxnorB)

A	101101
B	100100
fxn	101
X	110110



Test Vector 7. (A+B)

A	001111
B	111001
fxn	110
X	001000

Test Vector 8.(A-B)

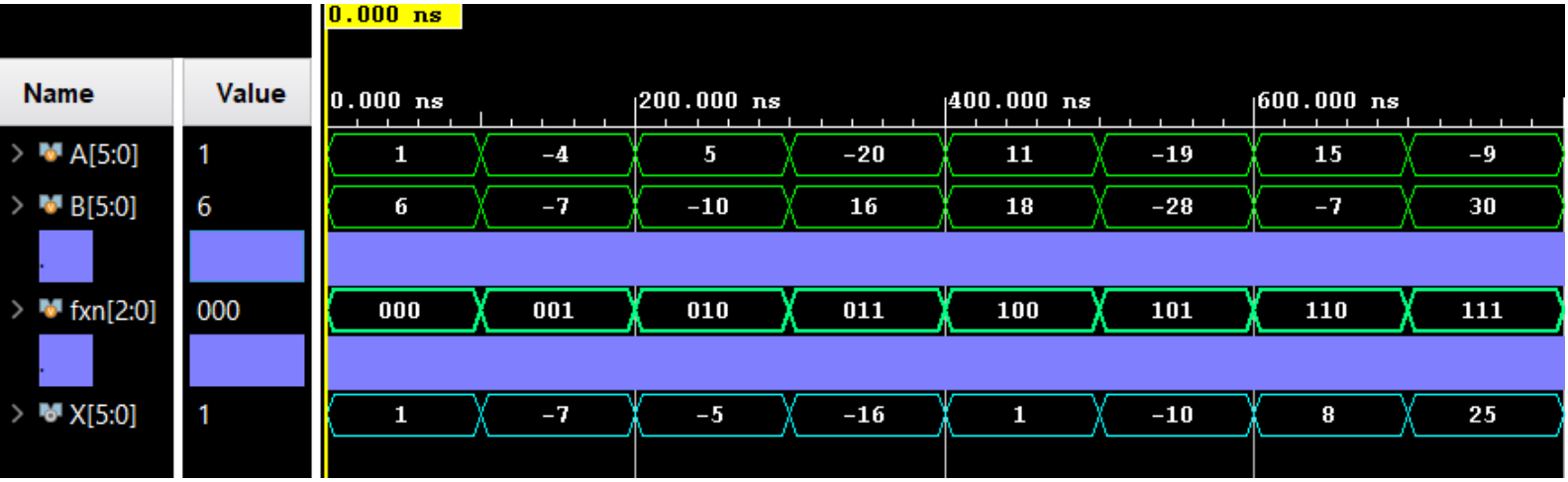
A	110111
B	011110
fxn	111
X	011001

APPENDIX

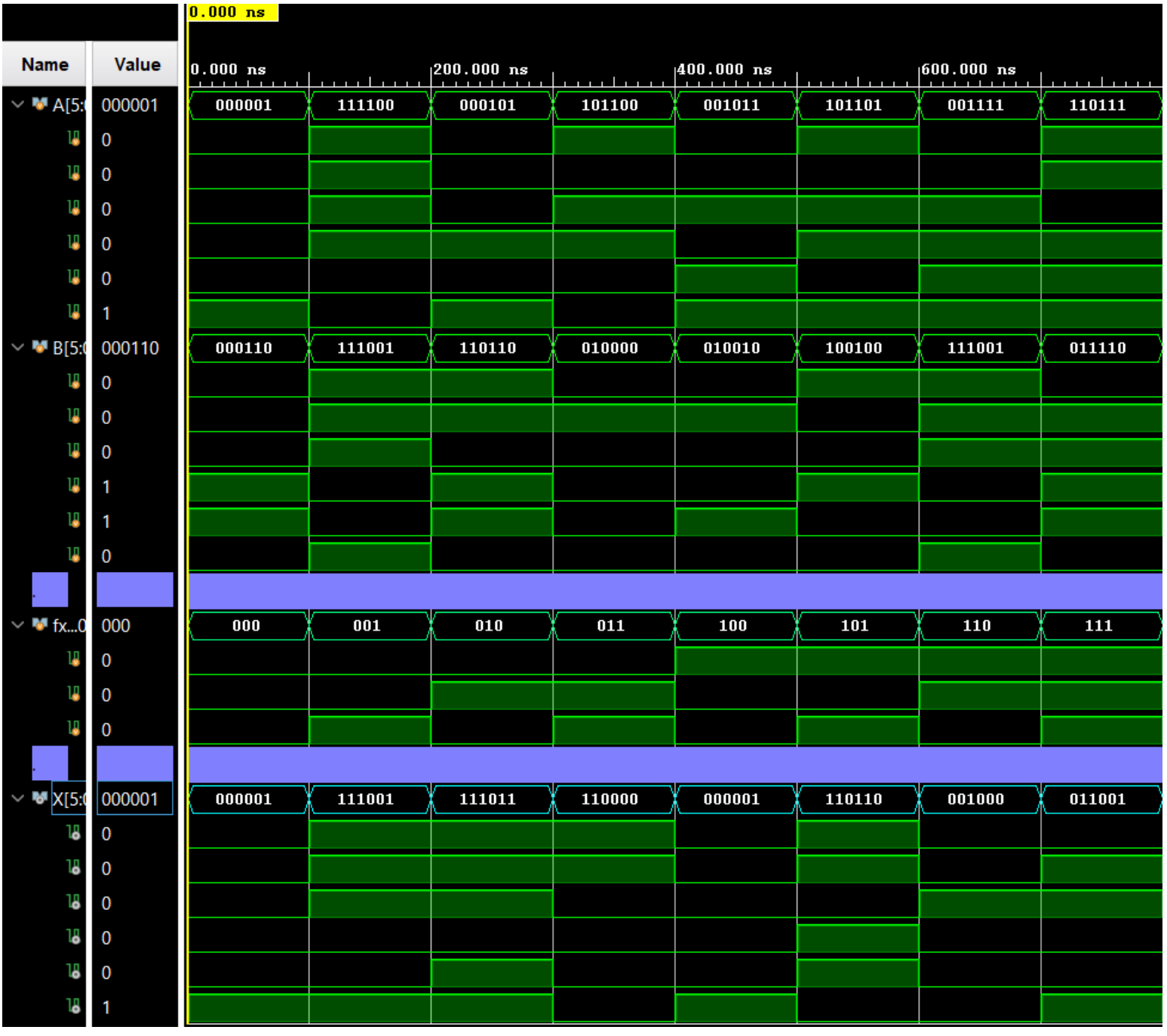
- **<u>fxn input controls for X output (Assignment 1 Description)</u>**

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

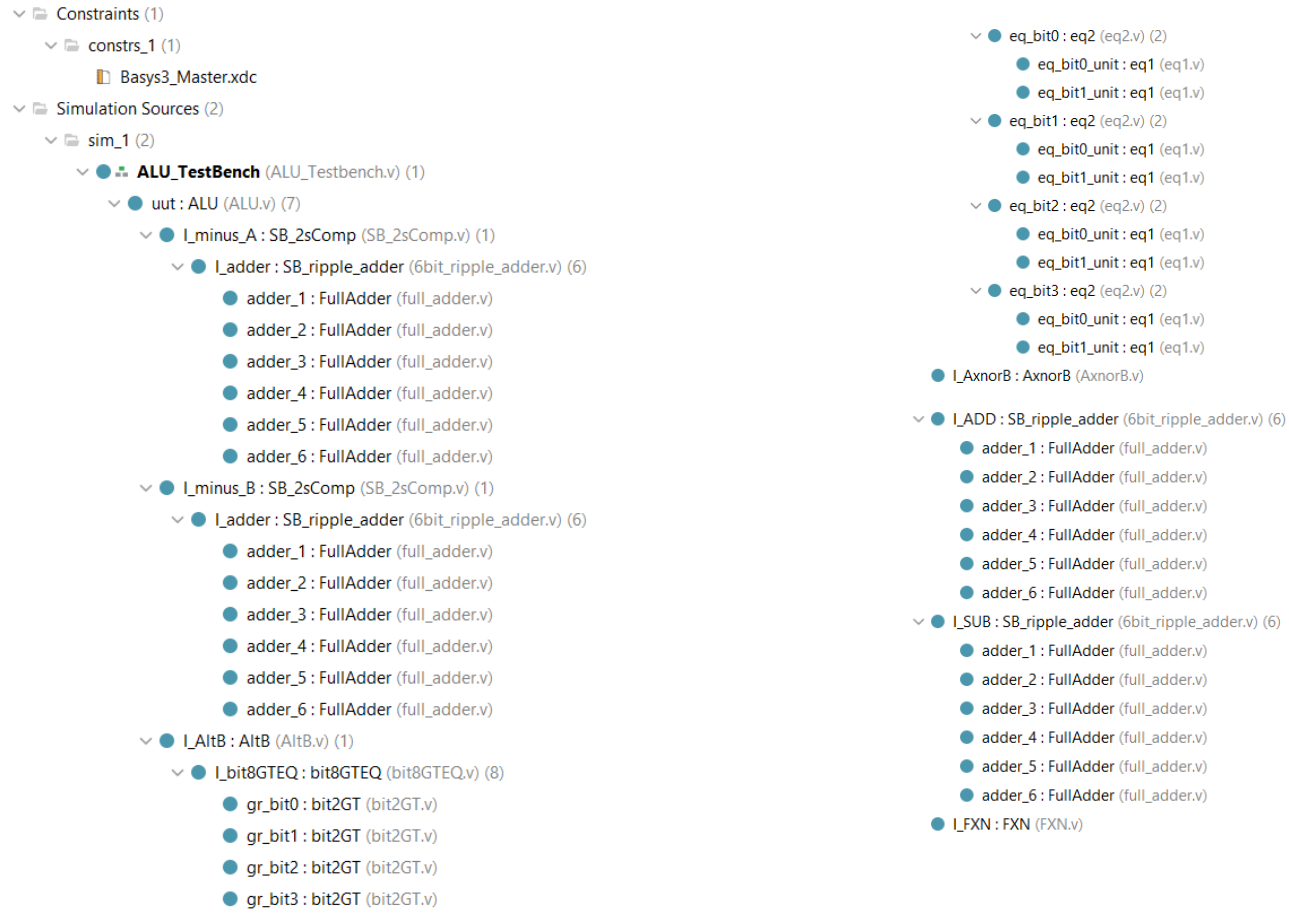
- **Waveform (Signed Decimal)**



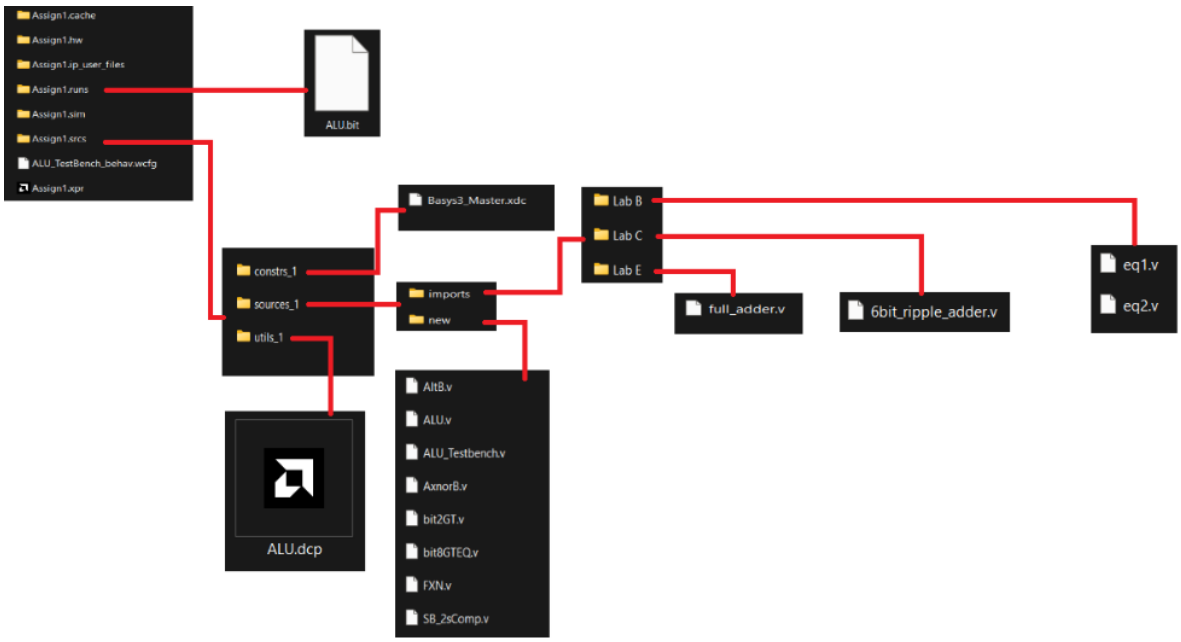
- **Waveform (Full)**



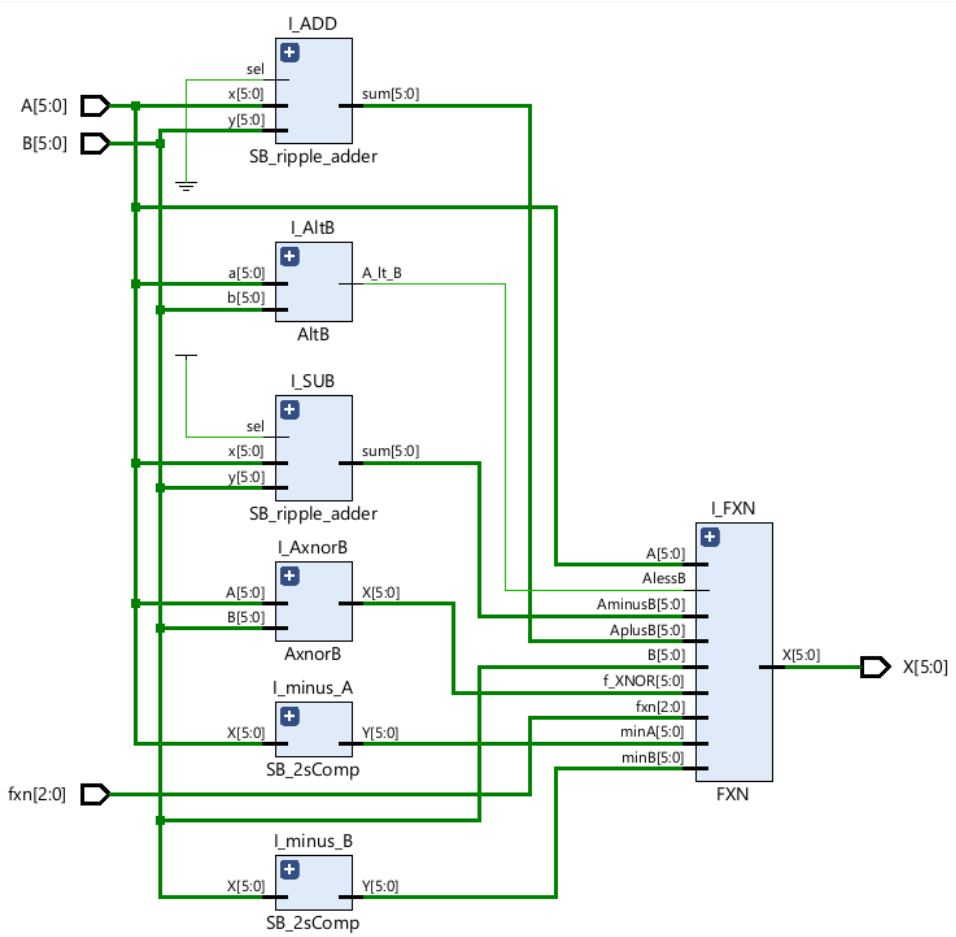
- **Vivado Hierarchy**



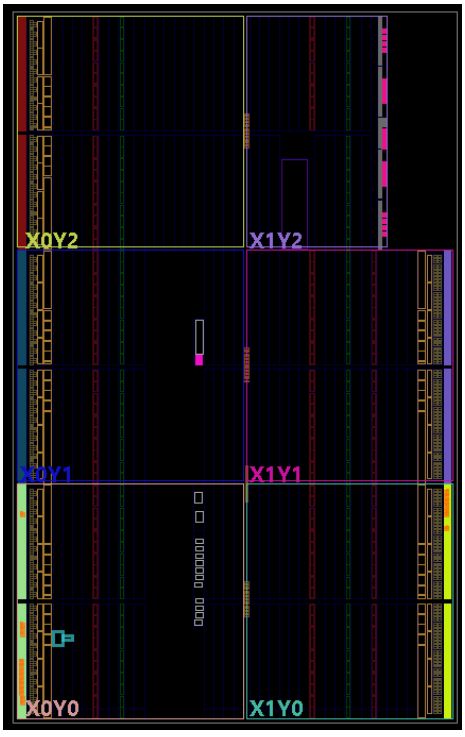
- **File Directory Structure**



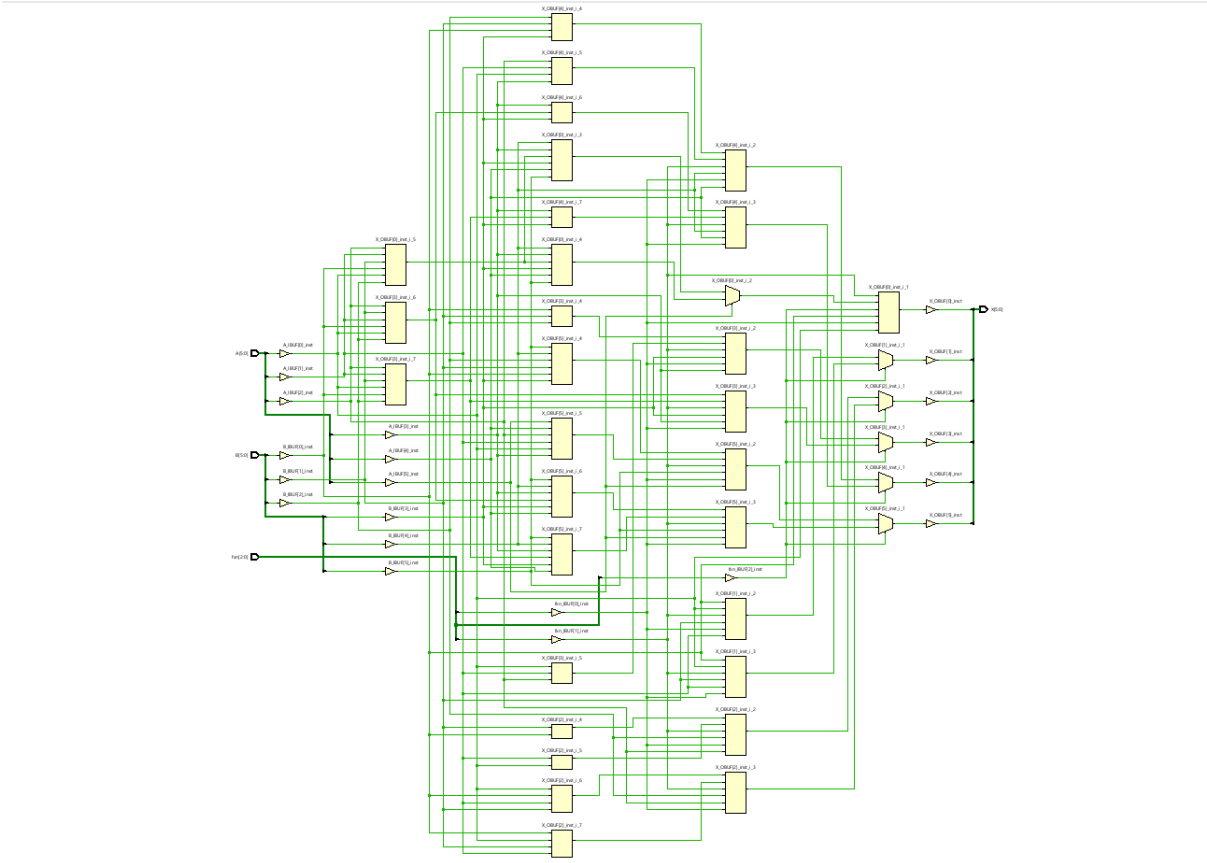
- **Vivado - Elaborate Design**



- **Vivado - Implemented Device**



- **Vivado – Synthesized Design Schematics**



- **Previous Submissions – Greater-than***

[*A greater-than report was not previously submitted, however Verilog code that would have been utilised in that lab was created in the form of bit8GTEQ.v and bit2GT.v while making the ALU for Assignment 1]

bit8GTEQ.v

```
module bit8GTEQ(
    input  wire[7:0] c, d,                  // 8-bit input vectors which represent A and B respectively
    output wire GTEQ                        // single bit output
    );
    wire e0, e1, e2, e3, g0, g1, g2, g3;      //e->equal | g->greater

    // bits 0-3 of 2-bit greater-than modules compare four sets of bits from 8-bit c and d
    bit2GT gr_bit0 (.a(c[1:0]), .b(d[1:0]), .AgrB(g0)); //set 1 (bit 1 to 2)
    bit2GT gr_bit1 (.AgrB(g1), .a(c[3:2]), .b(d[3:2])); //set 2 (bit 3 to 4)
    bit2GT gr_bit2 (.AgrB(g2), .a(c[5:4]), .b(d[5:4])); //set 3 (bit 5 to 6)
    bit2GT gr_bit3 (.AgrB(g3), .a(c[7:6]), .b(d[7:6])); //set 4 (bit 7 to 8)

    assign GT = g3 | e3 & g2 | e3 & e2 & g1 | e3 & e2 & e1 & g0;    // calculate greater-than result for 8-bit comparison

    // bits 0-3 of 2-bit equal-to modules compare four sets of bits from 8-bit c and d
    eq2 eq_bit0 (.a(c[1:0]), .b(d[1:0]), .aeqb(e0)); //set 1 (bit 1 to 2)
    eq2 eq_bit1 (.aeqb(e1), .a(c[3:2]), .b(d[3:2])); //set 2 (bit 3 to 4)
    eq2 eq_bit2 (.aeqb(e2), .a(c[5:4]), .b(d[5:4])); //set 3 (bit 5 to 6)
    eq2 eq_bit3 (.aeqb(e3), .a(c[7:6]), .b(d[7:6])); //set 4 (bit 7 to 8)

    assign EQ = e3 & e2 & e1 & e0;      // calculate equal-to result for 8-bit comparison

    assign GTEQ = EQ | GT;       // output final result [1 = c greater than or equal to d] [0 = c less than d]

endmodule
```

bit2GT.v

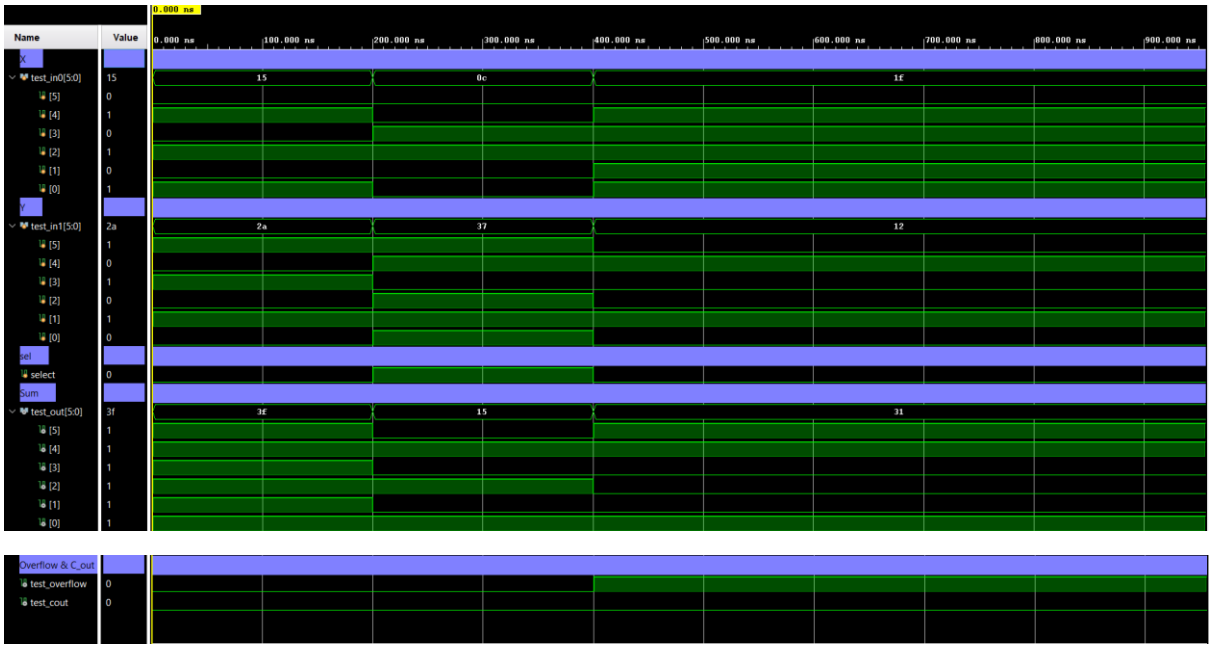
```
module bit2GT(
    input  wire[1:0] a, b,  // 2-bit inputs a and b
    output wire AgrB        // output shows if A greater than B
);

   wire [0:2] prod;         // 3 wires for product terms


   assign prod[0] = a[1] & ~b[1];             // prod-term0: a[1] AND (NOT b[1])
   assign prod[1] = a[0] & ~b[1] & ~b[0];     // prod-term1: a[0] AND (NOT b[1]) AND (NOT b[0])
   assign prod[2] = a[1] & a[0] & ~b[0];      // prod-term2: a[1] AND a[0] AND (NOT b[0])

   assign AgrB = prod[0] | prod[1] | prod[2]; // OR of all product terms to get output

endmodule
```

Previous Submissions – Adder(Lab C write up)

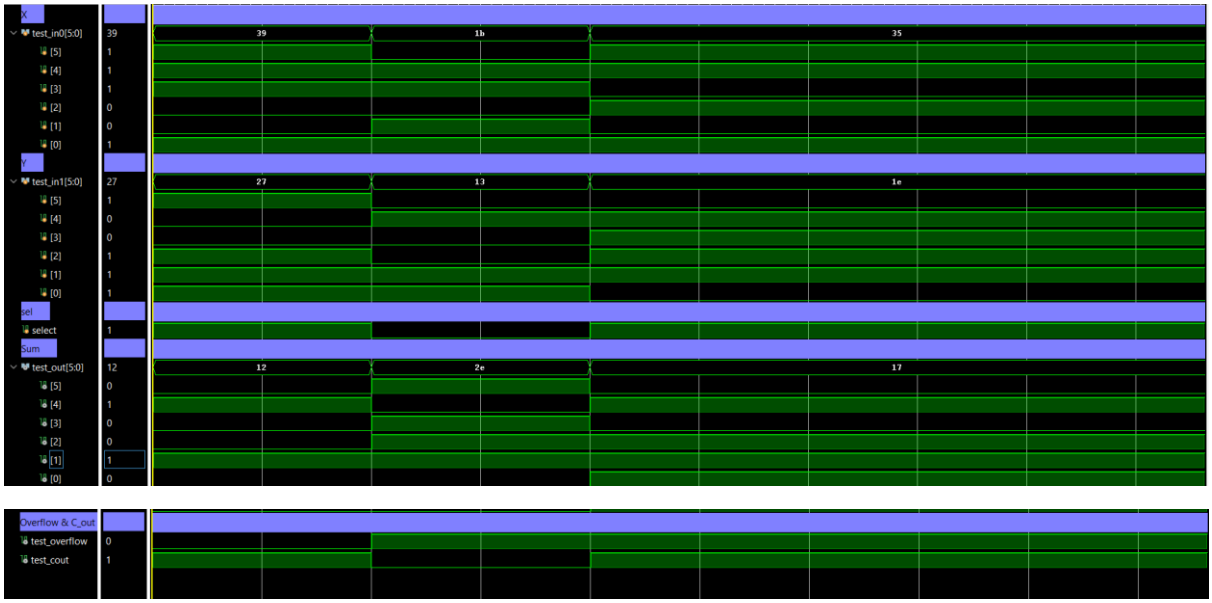
Test ID	x	y	sel	Cout (exp)	Overflow (exp)	Sum (exp)	Cout (obs)	Overflow (obs)	Sum (obs)	Pass/Fail
1	6'b010101	6'b101010	0	0	0	6'b111111	0	0	6'b111111	Pass
2	6'b001100	6'b110111	1	0	0	6'b010101	0	0	6'b010101	Pass
3	-6'b100001	6'b010010	0	0	1	6'b110001	0	1	6'b110001	Pass
4	6'b111001	6'b100111	1	1	0	6'b010010	1	0	6'b010010	Pass
5	6'b011011	-6'b101101	0	0	1	6’b101110	0	1	6’b101110	Pass
6	6'b110101	6’b011110	1	1	1	6’b010111	1	1	6’b010111	Pass
7	6’b111111	6’b111111	0	1	0	6’b111110	1	0	6’b111110	Pass
8	6’b000000	6’b000000	1	1	0	6’b000000	1	0	6’b000000	Pass
9	-6’b011010	-6’b101011	0	1	0	6’b111011	1	0	6’b111011	Pass

The circuit has passed all the tests as the observed output, overflow and sum values to what was expected. If I were to conduct this experiment again, I would increase the amount of test vectors with select as 1 in order to further verify the accuracy of the circuit when preforming 2's complement subtraction.

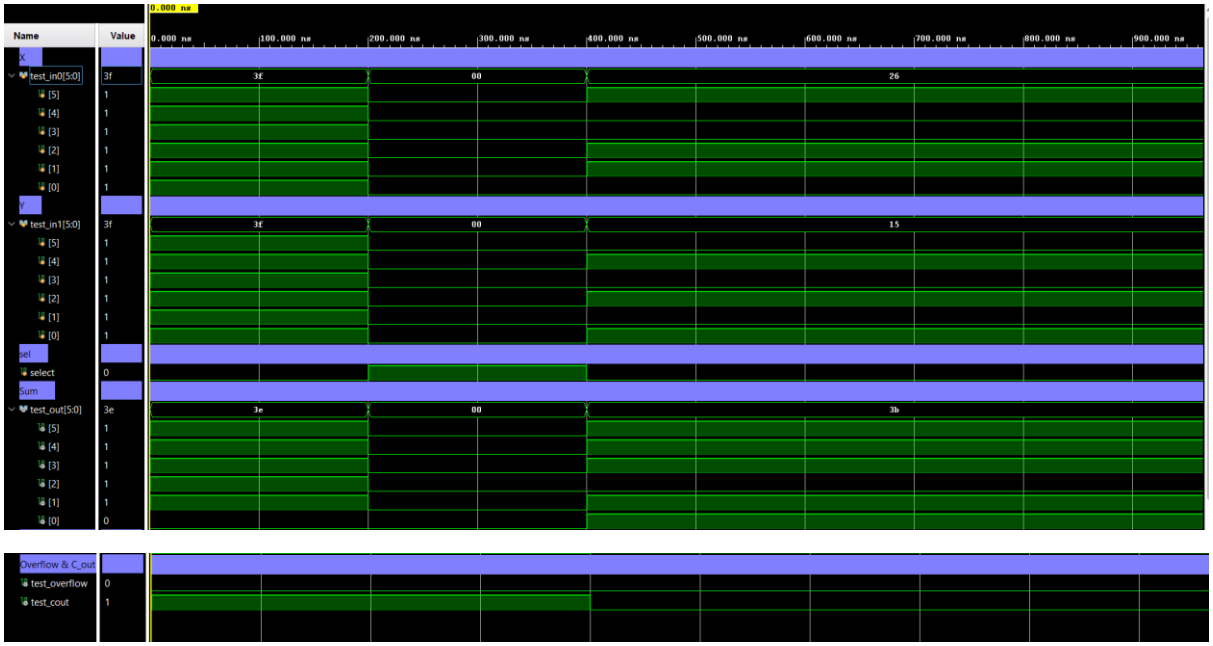
Testbench Waveform[1-3]



Testbench Waveform[3-6]



Testbench Waveform[6-9]



Block Diagram