Assignment 1

Name - Gwen McArdle

Student number – 18322248

Name of assignment – Mini Arithmetic Logic Unit

Date of submission – 17/11/2020

Aim

At the completion of this lab, I will have designed an ALU that allows the user chose between the following operations:

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

I will have used previously written, and new Verilog modules. I will have implemented the correct constraints and I will run the code on the FPGA boards. The switches provide 2's complement digits as inputs as well as the function choice input. The LEDs show the output in 2's complement.

Method

I designed a system, in which the inputs came from the switches and the outputs went to the LEDs via "Inputs_outputs" when using the board. When testing, the inputs came from "testbench", and the outputs were shown in the waveform.

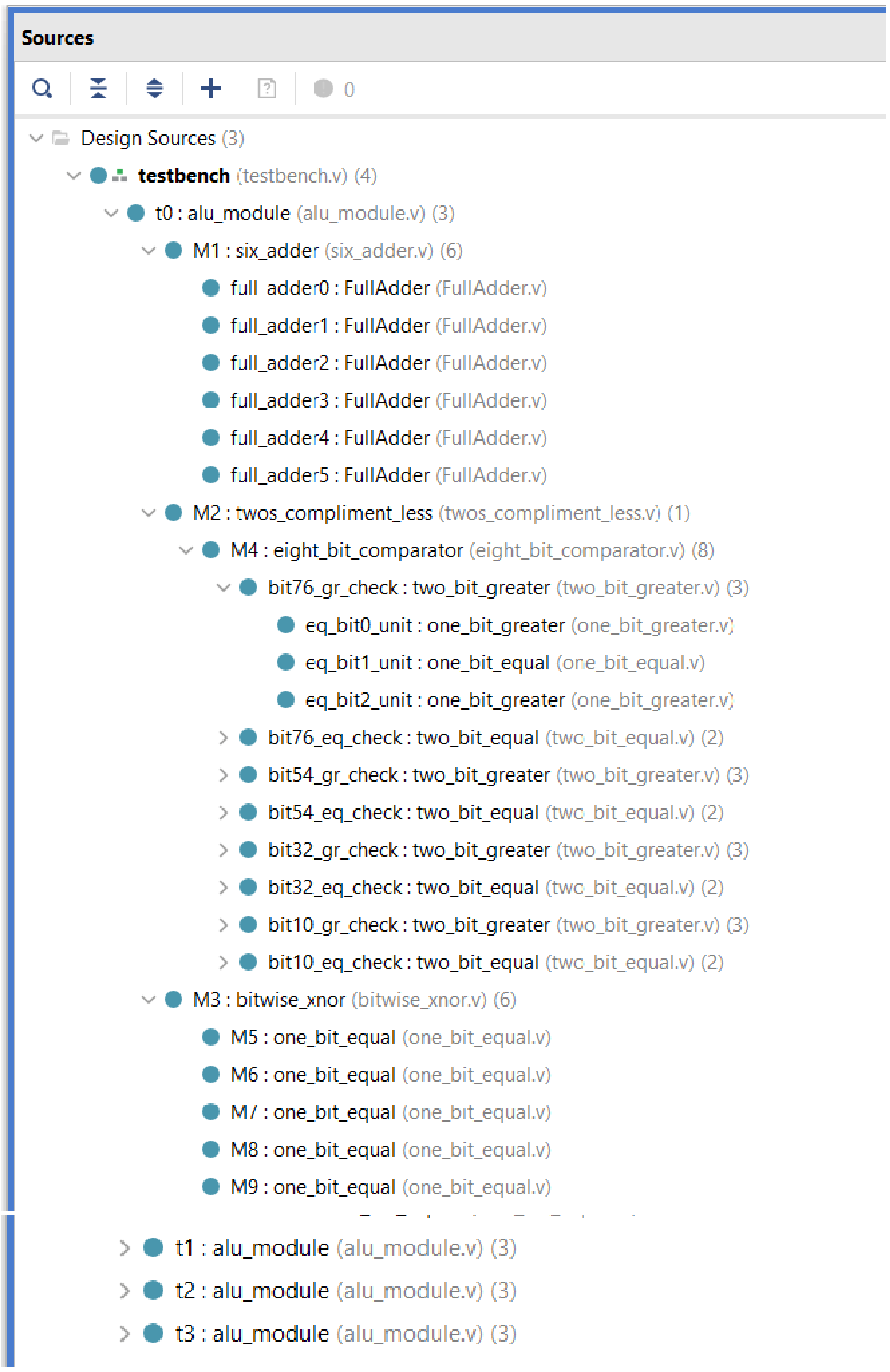
"alu_module" controls which operation's outputs are displayed on the LEDs. It implements "six_adder", "twos_compliment_less" and "bitwise_xnor". "six_adder" did not need any changes from Lab C.

"twos_compliment_less" is a module that adjusts it's inputs so they are in the correct form to implement "eight_bit_comparator". The inputs and outputs are also adjusted to accommodate the fact that the inputs are in two's compliment. I decided to add this module instead of making big changes to the comparator, because I wanted to preserve the original function of the comparator.

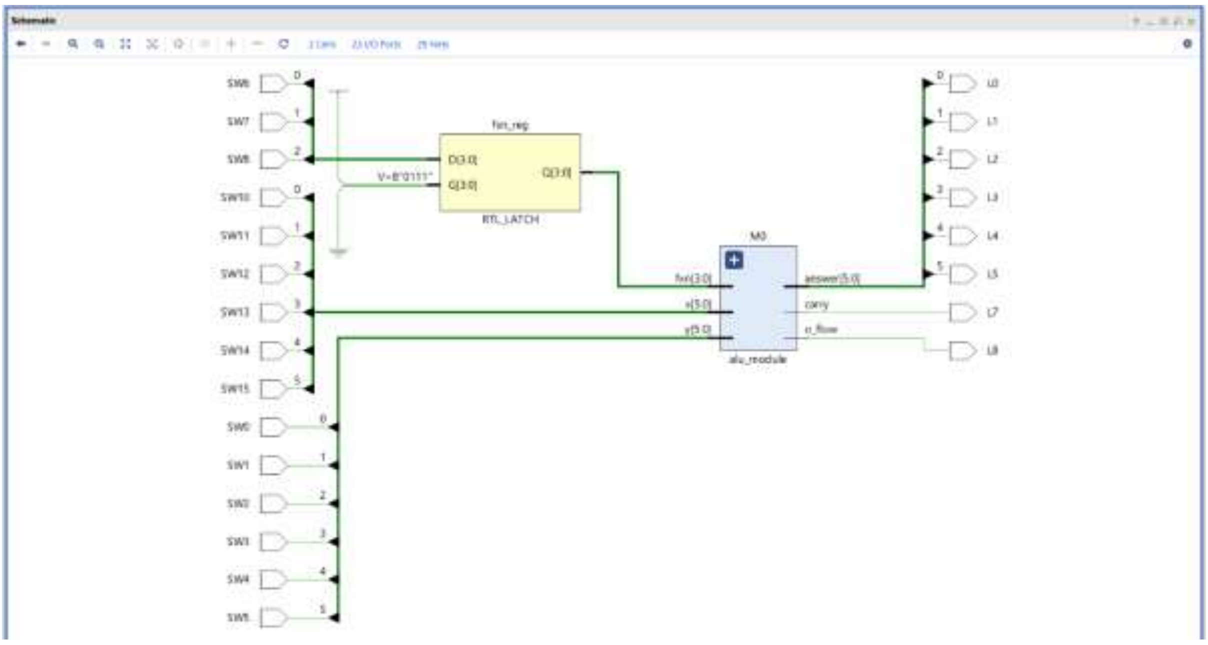
Even still, the "eight_bit_comparator" had to be adjusted slightly so that it returned whether or not they were greater or equal separately, this was necessary as the module is being used in two's complement so positive and negative numbers had to be dealt with differently.

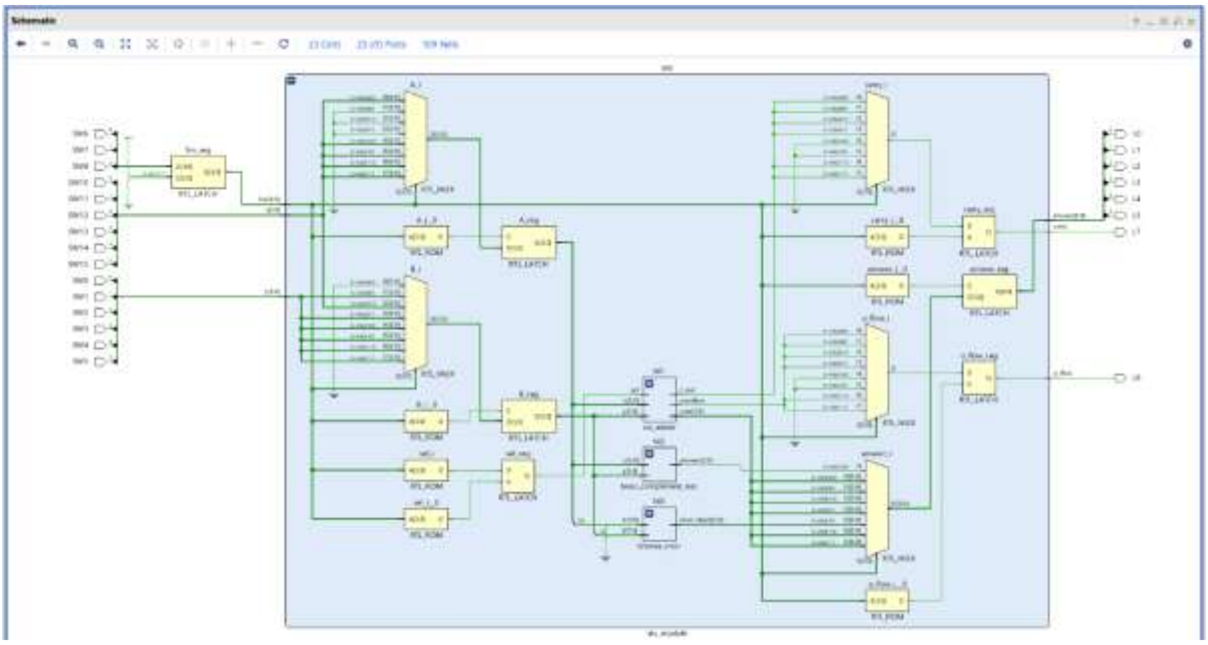
I wrote a simple module "bitwise_xnor" which implemented one_bit_equal (known as eq1 when provided).

The hierarchy is shown in the following screenshots:



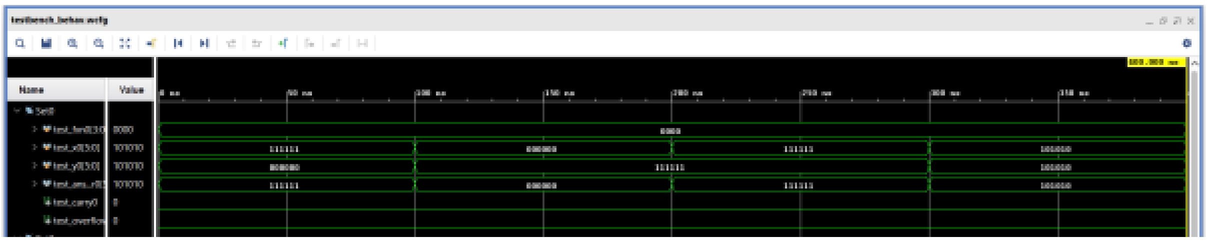
```
Sources

Design Sources (3)
  testbench (testbench.v) (4)
    t0 : alu_module (alu_module.v) (3)
      M1 : six_adder (six_adder.v) (6)
          full_adder0 : FullAdder (FullAdder.v)
          full_adder1 : FullAdder (FullAdder.v)
          full_adder2 : FullAdder (FullAdder.v)
          full_adder3 : FullAdder (FullAdder.v)
          full_adder4 : FullAdder (FullAdder.v)
          full_adder5 : FullAdder (FullAdder.v)
      M2 : twos_compliment_less (twos_compliment_less.v) (1)
        M4 : eight_bit_comparator (eight_bit_comparator.v) (8)
          bit76_gr_check : two_bit_greater (two_bit_greater.v) (3)
              eq_bit0_unit : one_bit_greater (one_bit_greater.v)
              eq_bit1_unit : one_bit_equal (one_bit_equal.v)
              eq_bit2_unit : one_bit_greater (one_bit_greater.v)
          bit76_eq_check : two_bit_equal (two_bit_equal.v) (2)
          bit54_gr_check : two_bit_greater (two_bit_greater.v) (3)
          bit54_eq_check : two_bit_equal (two_bit_equal.v) (2)
          bit32_gr_check : two_bit_greater (two_bit_greater.v) (3)
          bit32_eq_check : two_bit_equal (two_bit_equal.v) (2)
          bit10_gr_check : two_bit_greater (two_bit_greater.v) (3)
          bit10_eq_check : two_bit_equal (two_bit_equal.v) (2)
      M3 : bitwise_xnor (bitwise_xnor.v) (6)
          M5 : one_bit_equal (one_bit_equal.v)
          M6 : one_bit_equal (one_bit_equal.v)
          M7 : one_bit_equal (one_bit_equal.v)
          M8 : one_bit_equal (one_bit_equal.v)
          M9 : one_bit_equal (one_bit_equal.v)
    t1 : alu_module (alu_module.v) (3)
    t2 : alu_module (alu_module.v) (3)
    t3 : alu_module (alu_module.v) (3)
```


Schematic
2 Cells
23 I/O Ports
29 Nets
SW6
SW7
SW8
SW10
SW11
SW12
SW13
SW14
SW15
SW0
SW1
SW2
SW3
SW4
SW5
fxn_reg
D[3:0]
G[3:0]
Q[3:0]
V=B"0111"
RTL_LATCH
M0
fxn[3:0]
x[5:0]
y[5:0]
answer[5:0]
carry
o_flow
alu_module
L0
L1
L2
L3
L4
L5
L7
L8

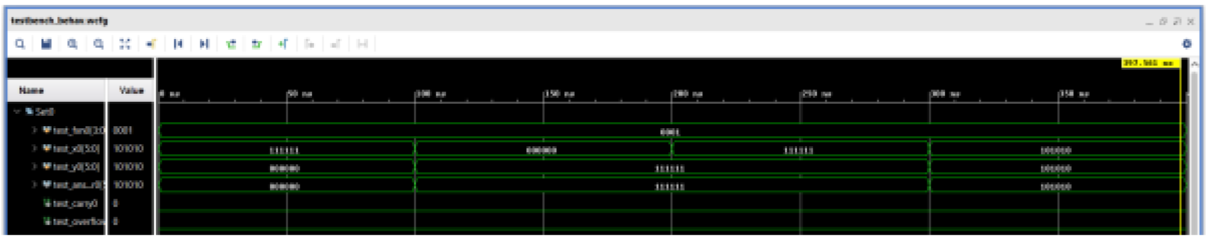

Schematic
23 Cells
23 I/O Ports
109 Nets
M0
fxn_reg
RTL_LATCH
A_i
RTL_MUX
A_reg
RTL_LATCH
B_i
B_reg
RTL_ROM
carry_i
carry_reg
answer_reg
o_flow_i
o_flow_reg
answer_i
alu_module

Results

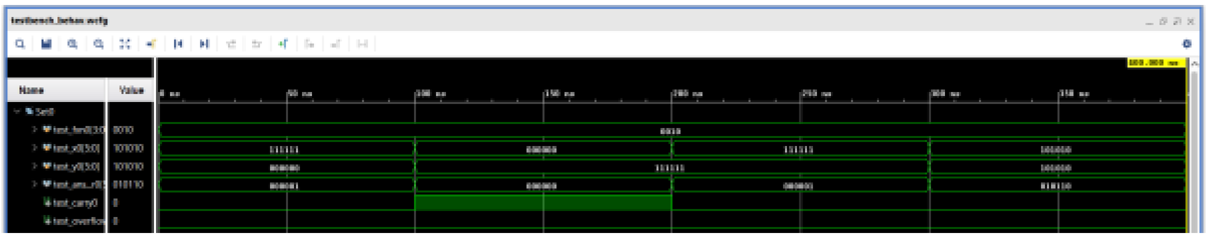
This waveform shows results when fxn = 000. The output should be input 1, which is the case.



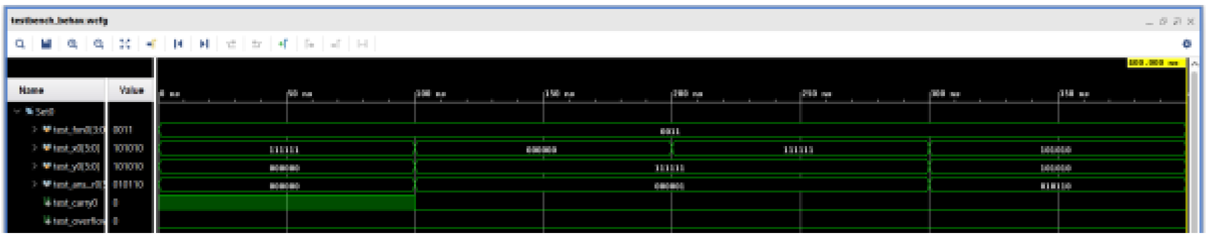
This waveform shows results when fxn = 001. The output should be input 2, which is the case.



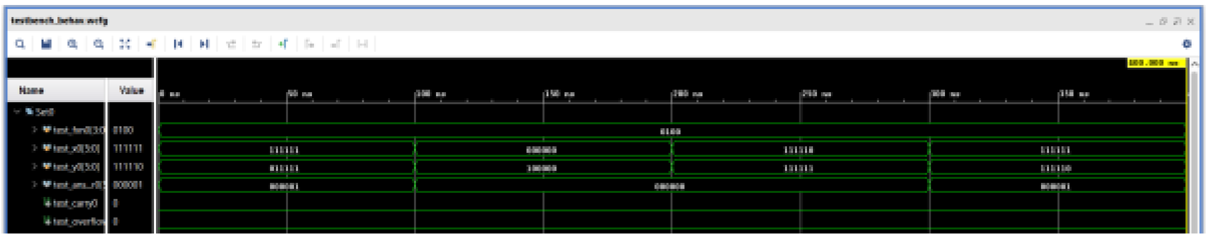
This waveform shows results when fxn = 010. The output should be the inverse of input 1, which is the case.



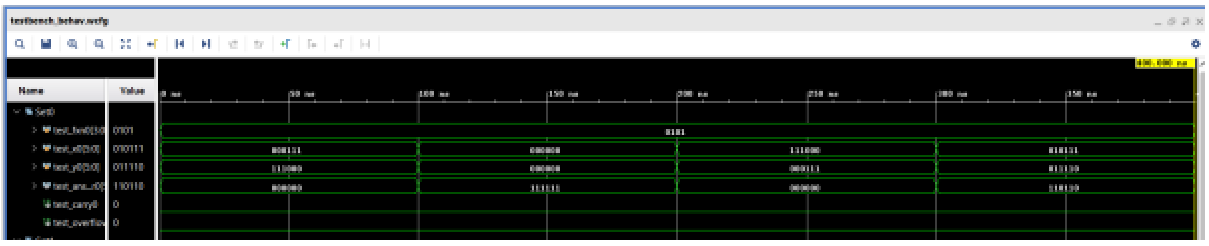
This waveform shows results when fxn = 011. The output should be the inverse of input 2, which is the case.



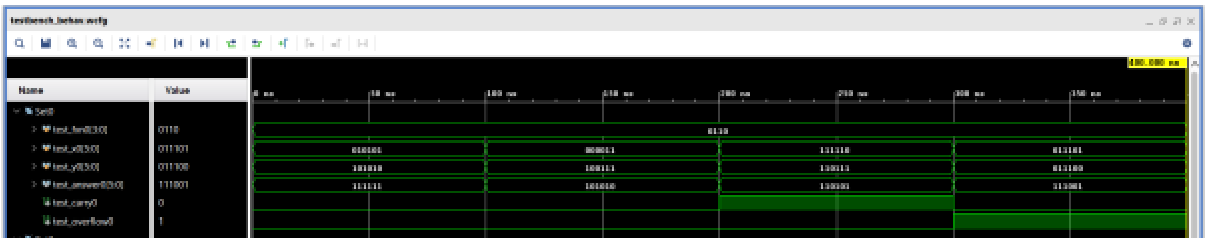
This waveform shows results when fxn = 100. The output should be 1 if input 1 is less than input 2, which is the case.



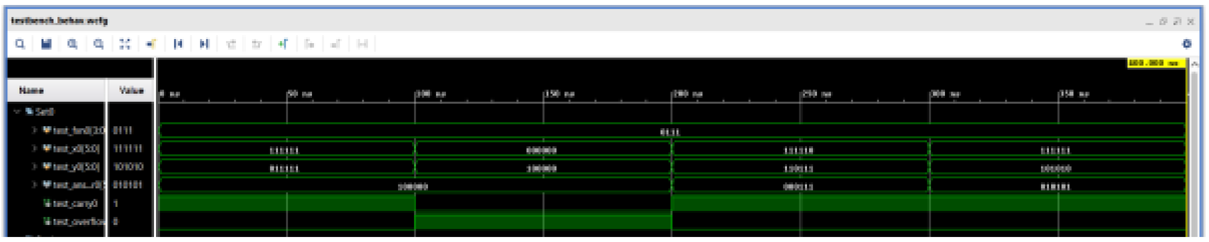
This waveform shows results when fxn = 101. Each bit of the output should be 1 if the corresponding bits from input 1 and input 2 are equal, which is the case.



This waveform shows results when fxn = 110. The output should be the sum of input 1 and input 2, which is the case.



This waveform shows results when fxn = 111. The output should be input 1 minus input 2, which is the case.



Note: In the submitted file, all of these are in the same waveform, each column is the same function.

Instructions for demonstration

In order to demo my ALU:

1. The submitted code uses the module "inputs_outputs" in the file main to control the system.
2. Generate the bitstream and program the device.
3. Use switches 10-15 for input 1 (A).
4. Use switches 6-8 for choose the function (fxn).
5. Use switches 0-5 for input 2 (B).
6. In the case that you want to run the testbench, disable "main", enable "testbench" and run the simulation.

Discussion of tests

The mini ALU is quite effective at meeting the aims of this assignment. Each function was strategically tested, so that any errors could be identified.

fxn = 000, was tested with inputs A and B, such that it tested whether it outputted A when there was a conflicting input from B, and when there was a matching input from B. For each combination of inputs the A was always outputted. Therefore, no limitations were identified when fxn = 000.

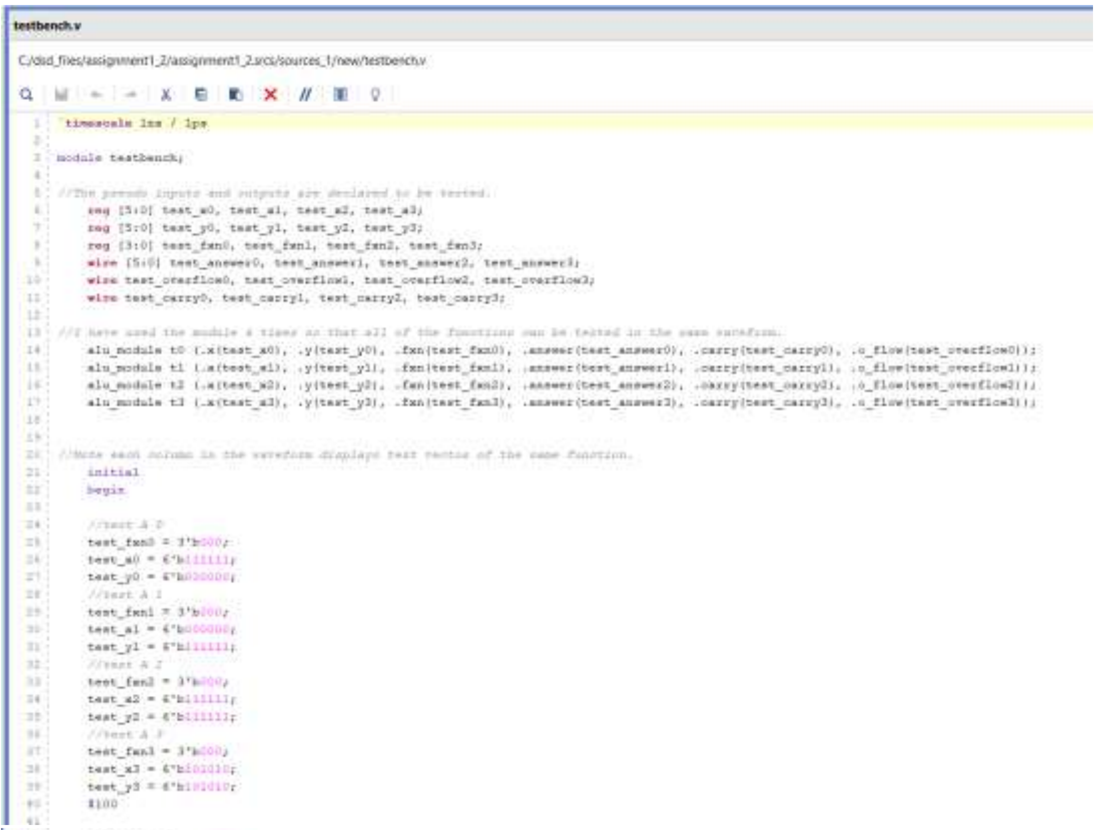
fxn = 001, was tested with the same inputs as fxn = 000, as the function was largely the same, except this time B was outputted. This test was passed for each combination of inputs.

The same inputs were used to test fxn = 010 and fxn = 011. Again, these functions are quite similar to the first two. However, these functions must also be checked for how they handle inputs that yield a carry when they are converted to a negative two's compliment number. Both functions, gave the correct outputs for each test.

fxn = 100 had slightly different test vectors. "twos_compliment_less" has multiple cases. Firstly when the MSB of A is 0 and the MSB of B is 1, the output should be one because negative numbers are always less than positive numbers, this test was passed. When the MSB of A is 1 and the MSB of B is 0, the opposite should be true, the module was also successful when given these inputs. However, when the MSB of each input is equal, bits 0 to 4 of the two inputs must be compared. If they are positive numbers then 1 should be outputted if A is not greater than nor equal to B. These tests were passed. If they are negative numbers then 1 should be outputted if A is greater than and not equal to B. Unfortunately, when tested with two equal negative numbers, 1 was outputted. This is not included in the waveform as it was tested with the switches.

fxn = 101 and fxn = 111 were tested with a variety of inputs, including those that should yield carry and overflow. Both of these functions gave the correct results for each test.

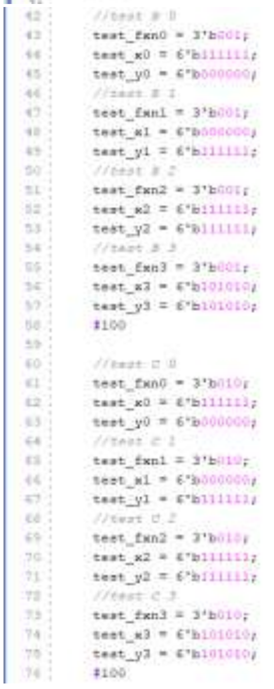
The following pages include screenshots of the testbench, but additional testing was undertaken using the switches on the board.



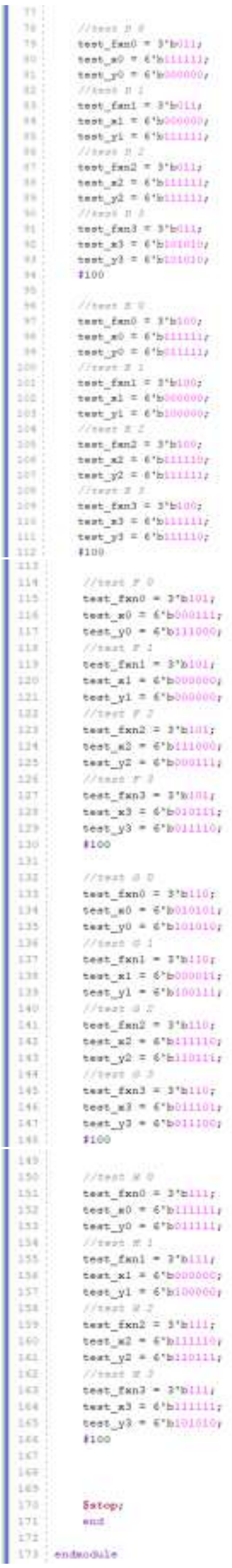
testbench.v

C:/dsd_files/assignment1_2/assignment1_2.srcs/sources_1/new/testbench.v

```verilog
`timescale 1ns / 1ps

module testbench;

//The pseudo inputs and outputs are declared to be tested.
    reg [5:0] test_x0, test_x1, test_x2, test_x3;
    reg [5:0] test_y0, test_y1, test_y2, test_y3;
    reg [3:0] test_fxn0, test_fxn1, test_fxn2, test_fxn3;
    wire [5:0] test_answer0, test_answer1, test_answer2, test_answer3;
    wire test_overflow0, test_overflow1, test_overflow2, test_overflow3;
    wire test_carry0, test_carry1, test_carry2, test_carry3;

//I have used the module 4 times so that all of the functions can be tested in the same waveform.
    alu_module t0 (.x(test_x0), .y(test_y0), .fxn(test_fxn0), .answer(test_answer0), .carry(test_carry0), .o_flow(test_overflow0));
    alu_module t1 (.x(test_x1), .y(test_y1), .fxn(test_fxn1), .answer(test_answer1), .carry(test_carry1), .o_flow(test_overflow1));
    alu_module t2 (.x(test_x2), .y(test_y2), .fxn(test_fxn2), .answer(test_answer2), .carry(test_carry2), .o_flow(test_overflow2));
    alu_module t3 (.x(test_x3), .y(test_y3), .fxn(test_fxn3), .answer(test_answer3), .carry(test_carry3), .o_flow(test_overflow3));


//Note each column in the waveform displays test vector of the same function.
    initial
    begin

    //test A 0
    test_fxn0 = 3'b000;
    test_x0 = 6'b111111;
    test_y0 = 6'b000000;
    //test A 1
    test_fxn1 = 3'b000;
    test_x1 = 6'b000000;
    test_y1 = 6'b111111;
    //test A 2
    test_fxn2 = 3'b000;
    test_x2 = 6'b111111;
    test_y2 = 6'b111111;
    //test A 3
    test_fxn3 = 3'b000;
    test_x3 = 6'b101010;
    test_y3 = 6'b101010;
    #100

```



```verilog
    //test B 0
    test_fxn0 = 3'b001;
    test_x0 = 6'b111111;
    test_y0 = 6'b000000;
    //test B 1
    test_fxn1 = 3'b001;
    test_x1 = 6'b000000;
    test_y1 = 6'b111111;
    //test B 2
    test_fxn2 = 3'b001;
    test_x2 = 6'b111111;
    test_y2 = 6'b111111;
    //test B 3
    test_fxn3 = 3'b001;
    test_x3 = 6'b101010;
    test_y3 = 6'b101010;
    #100

    //test C 0
    test_fxn0 = 3'b010;
    test_x0 = 6'b111111;
    test_y0 = 6'b000000;
    //test C 1
    test_fxn1 = 3'b010;
    test_x1 = 6'b000000;
    test_y1 = 6'b111111;
    //test C 2
    test_fxn2 = 3'b010;
    test_x2 = 6'b111111;
    test_y2 = 6'b111111;
    //test C 3
    test_fxn3 = 3'b010;
    test_x3 = 6'b101010;
    test_y3 = 6'b101010;
    #100
```



```
    //test D 0
    test_fxn0 = 3'b011;
    test_x0 = 6'b111111;
    test_y0 = 6'b000000;
    //test D 1
    test_fxn1 = 3'b011;
    test_x1 = 6'b000000;
    test_y1 = 6'b111111;
    //test D 2
    test_fxn2 = 3'b011;
    test_x2 = 6'b111111;
    test_y2 = 6'b111111;
    //test D 3
    test_fxn3 = 3'b011;
    test_x3 = 6'b101010;
    test_y3 = 6'b101010;
    #100

    //test E 0
    test_fxn0 = 3'b100;
    test_x0 = 6'b111111;
    test_y0 = 6'b011111;
    //test E 1
    test_fxn1 = 3'b100;
    test_x1 = 6'b000000;
    test_y1 = 6'b100000;
    //test E 2
    test_fxn2 = 3'b100;
    test_x2 = 6'b111110;
    test_y2 = 6'b111111;
    //test E 3
    test_fxn3 = 3'b100;
    test_x3 = 6'b111111;
    test_y3 = 6'b111110;
    #100

    //test F 0
    test_fxn0 = 3'b101;
    test_x0 = 6'b000111;
    test_y0 = 6'b111000;
    //test F 1
    test_fxn1 = 3'b101;
    test_x1 = 6'b000000;
    test_y1 = 6'b000000;
    //test F 2
    test_fxn2 = 3'b101;
    test_x2 = 6'b111000;
    test_y2 = 6'b000111;
    //test F 3
    test_fxn3 = 3'b101;
    test_x3 = 6'b010111;
    test_y3 = 6'b011110;
    #100

    //test G 0
    test_fxn0 = 3'b110;
    test_x0 = 6'b010101;
    test_y0 = 6'b101010;
    //test G 1
    test_fxn1 = 3'b110;
    test_x1 = 6'b000011;
    test_y1 = 6'b100111;
    //test G 2
    test_fxn2 = 3'b110;
    test_x2 = 6'b111110;
    test_y2 = 6'b110111;
    //test G 3
    test_fxn3 = 3'b110;
    test_x3 = 6'b011101;
    test_y3 = 6'b011100;
    #100

    //test H 0
    test_fxn0 = 3'b111;
    test_x0 = 6'b111111;
    test_y0 = 6'b011111;
    //test H 1
    test_fxn1 = 3'b111;
    test_x1 = 6'b000000;
    test_y1 = 6'b100000;
    //test H 2
    test_fxn2 = 3'b111;
    test_x2 = 6'b111110;
    test_y2 = 6'b110111;
    //test H 3
    test_fxn3 = 3'b111;
    test_x3 = 6'b111111;
    test_y3 = 6'b101010;
    #100



    $stop;
    end

endmodule
```

Appendices

The following reports have previously been submitted:

LabA_mcardleg.pdf

LabB_mcardleg.pdf

LabC_mcardleg.pdf