CprE 3810, Computer Organization and

Assembly Level Programming

Lab 1 Report

Student Name **Connor Moroney**

Submit a typeset pdf version of this on Canvas by the due date. Refer to the highlighted language in the lab document for the context of the following questions.

[Part 3.1.c] Think of three more cases and record them in your lab report.

- **Case 3: Perform a MAC operation that results in oY being the same value as the day of my birth.**
  - **I was born on the 21st of March. Therefore, I need to adjust my input to make the oY output 21 (The same day I was born)**
  - **I will set iW to 3, iLdW will need to be set to 1, iX should be set to 7 and iY should be set to 0.**
  - **I expect that, after two positive edges of the clock (i.e., two cycles), the activation output, oX, will be 7 and the partial sum output, oY will be 21=3*7+0.**
- **Case 4: Confirm that iW doesn't change unless iLdW is set to 1,**
  - **Assuming that iW is still 3 from the previous test, I will set iW to 20, but I will set iLdW to 0 which will mean that the value of iW doesn't change (it should still be 3).**
  - **I will set oX to 15 and oY to 5.**
  - **I expect that, after two positive edges of the clock (i.e., two cycles), the activation output, oX, will be 15 and the partial sum output, oY will be 50=3*15+5.**
- **Case 5: Set all inputs to 0**
  - **I will set oX to 0, oY to 0, iLdW to 0, and iW to 0.**
  - **I expect that, after two positive edges of the clock (i.e., two cycles), the activation output, oX, will be 0 and the partial sum output, oY will be 0=3*0+0.**

[Part 3.1.e] For labels 1, 7, 22, and 28, specify where (VHDL file and line number) these values are located – some will be found in more than one place. Also attempt to explain the functionality of each label as it occurs in the code

In the attached diagram area, (1) is TPU_MV_Element, defined on Line 24 of TPU_MV_Element.vhd, it contains all of the input signals (iCLK, iX, iW, iLdW, iY) and output signals (oY and oX). As seen on Line 38, the TPU_MV_Element also contains all of the subcomponents that are used in the calculation. Therefore, TPU_MV_Element is the main component.

In the attached diagram area, (7) is g_Add1, defined on Line 131 of TPU_MV_Element.vhd as an adder module. This module has three inputs (iA, iB, and iCLK) and one output (oC). On Line 45 of Adder.vhd, we can see that on the rising edge of the clock, the output oC is the sum of the two main inputs (iA and iB)

In the attached diagram area, (22) is the oQ output of the weight module, which is defined as a DATA_WIDTH-bit std_logic_vector on Line 30 of RegLd.vhd and set on Line 50. For this specific instance of the weight module, the output oQ is connected to a signal called s_W on line 96 of TPU_MV_Element.vhd.

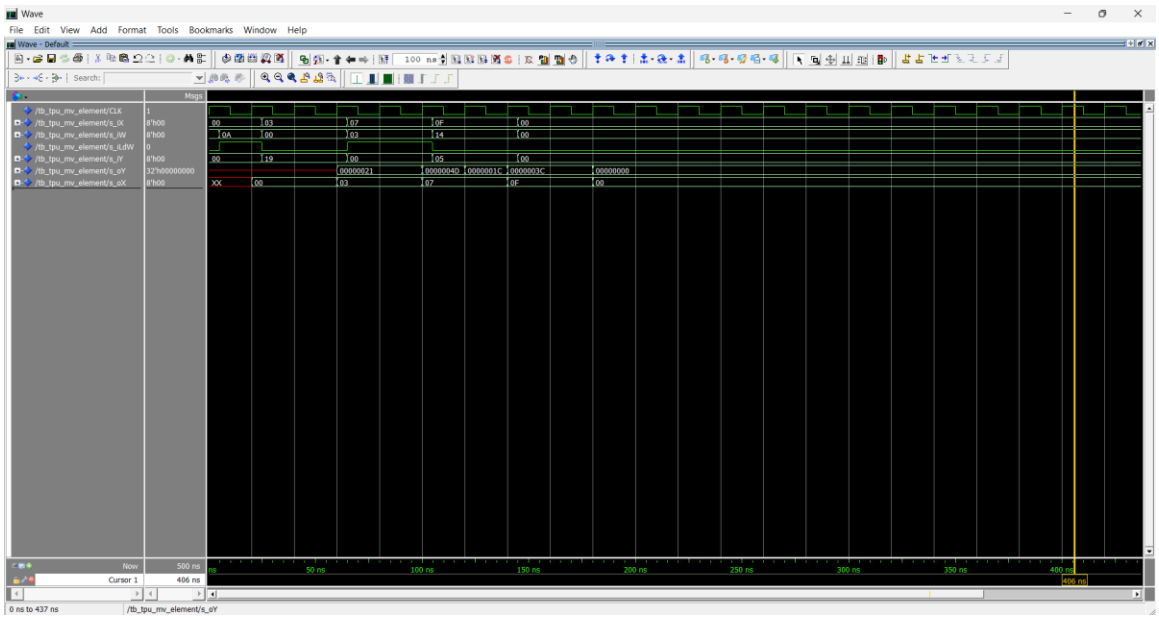
In the attached diagram area, (28) is the iD input of the delay module, which is defined as a in DATA_WIDTH-bit std_logic_vector on Line 28 of Reg.vhd and used to set oQ on Line 39. For this specific instance of the delay module, the input iD is connected to a signal called s_X1 on line 128 of TPU_MV_Element.vhd.

[Part 3.1.f] Implement your test cases from part c into this testbench. Remember to include these in your waveform screenshot.

```
126     -- Test case 3:
127     -- Perform a MAC operation that results in oY being the same value as the day of my birth (21).
128     s_iX   <= x"07";  -- 7 in hexadecimal
129     s_iW   <= x"03";  -- 3 in hexadecimal
130     s_iLdW <= '1';    -- Load so that iW will change
131     s_iY   <= x"00";  -- 0 in hexadecimal
132     wait for gCLK_HPER*2;
133     wait for gCLK_HPER*2;
134     -- Expect: o_Y output signal to be 21 = 7*3+0 and o_X output signal to be 7 after two positive edge of clock.
135
136
137     -- Test case 4:
138     -- Confirm that iW doesn't change unless iLdW is set to 1
139     s_iX   <= x"0E";  -- 15 in hexadecimal
140     s_iW   <= x"14";  -- 20 in hexadecimal
141     s_iLdW <= '0';    -- Make sure we don't continue to load.
142     s_iY   <= x"05";  -- 5 in hexadecimal
143     wait for gCLK_HPER*2;
144     wait for gCLK_HPER*2;
145     -- Expect: o_Y output signal to be 50 = 15*3+5 and o_X output signal to be 15 after two positive edge of clock.
146
147     -- Test case 5:
148     -- Set all inputs to 0
149     s_iX   <= x"00";  -- 0 in hexadecimal
150     s_iW   <= x"00";  -- 0 in hexadecimal
151     s_iLdW <= '0';    -- Make sure we don't continue to load.
152     s_iY   <= x"00";  -- 0 in hexadecimal
153     wait for gCLK_HPER*2;
154     wait for gCLK_HPER*2;
155     -- Expect: o_Y output signal to be 0 = 0*3+0 and o_X output signal to be 0 after two positive edge of clock.
156
```

Note that I slightly changed test 3 after debugging (Final test can be seen in the final waveform

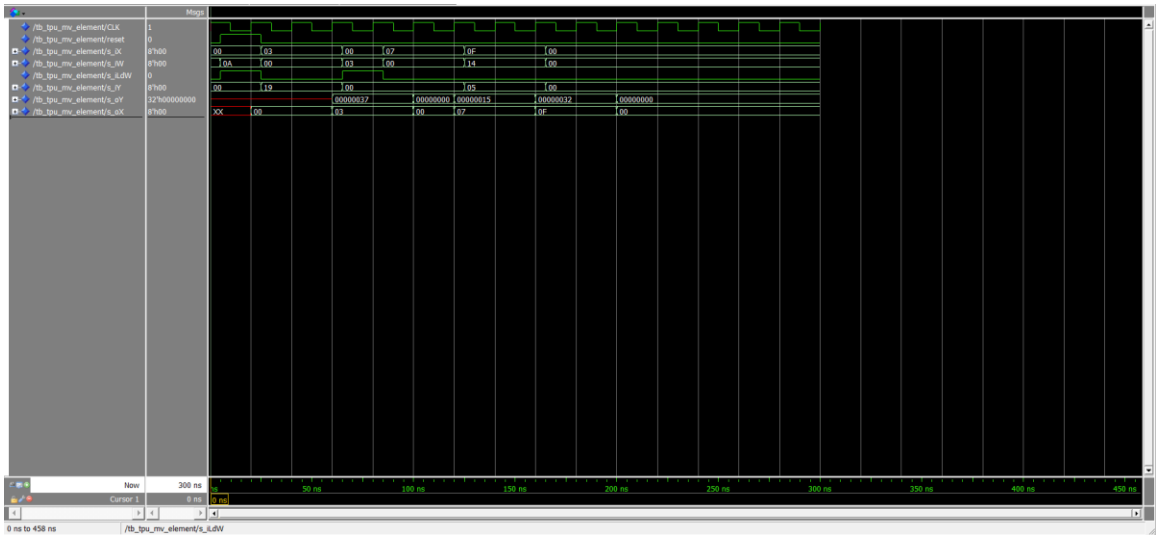
[Part 3.1.g.v] In your lab report, include a screenshot of the waveform. Describe, in plain English, any differences between what you expected and what the simulation showed.



In the second test, we expected oX to be 3 and oY to be 55. Although oX matched the expected, oY was 33 instead of 55. In the third test, we see a problem with two outputs**. Again, oX is correct, but oY is incorrect. oX is correct for test 4, but oY is incorrect once again. The 5th test matched the expected results (oX and oY were both 0).

**This problem stems from my misunderstanding of how this code works when I was designing the test cases

[Part 3.1.h] In your lab report, include a screenshot of the waveform. Describe, in plain English, how your waveform matches the expected result (e.g., reference the specific cycles and times). In your submission zip file, provide the completed *TPU_MV_Element.vhd* file in a folder called 'MAC'.



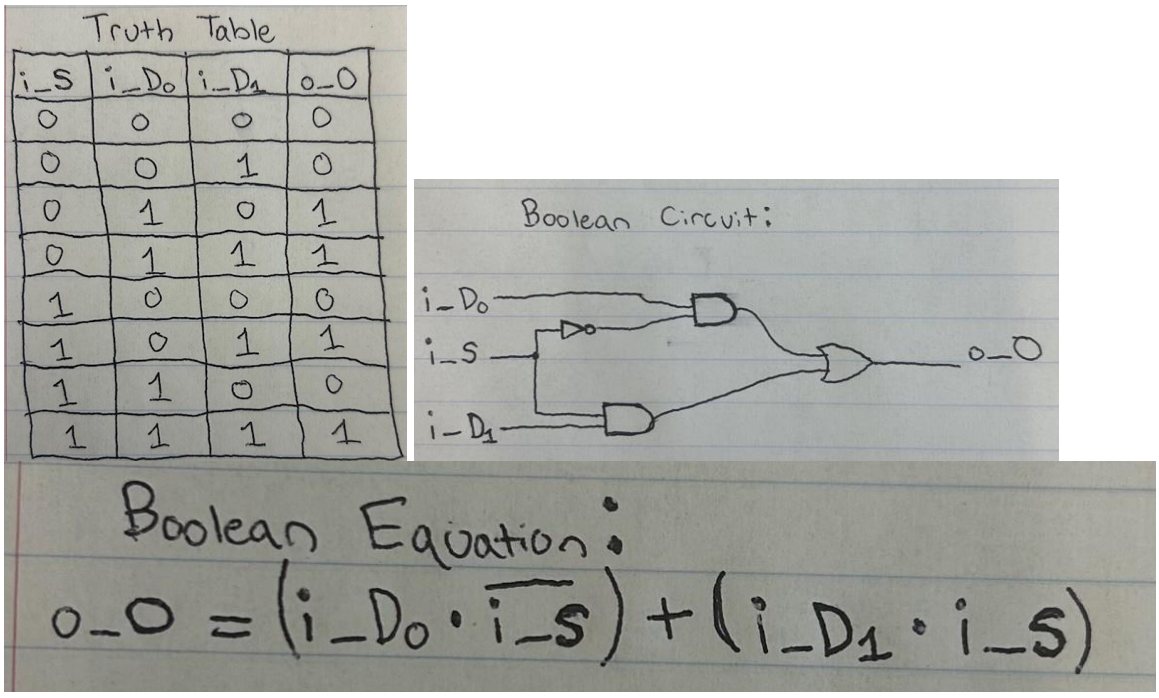
In the first test, we can see that iW is being set to 10 and that iLdW is high. oX is 0 as expected and there is no output for oY. This means that the first test case works as expected. In the second test, we can see that oX is 03 and oY is 55, this matched the

expected output. In the third test, oX and oY are 0 while iW is loaded, and then 7 and 21 respectively. Again, matching the expected results. For test 4, oX is 15 and oY is 50, which matches the expected results. And in test 5, when all inputs are set to 0, both outputs are 0 (which is expected).

In order to fix the output to match expected, on top of the bug fixes, I also needed to update my 3rd Test case. Essentially I needed to set iX and iY to 0 when loading iW to prevent a confusing output where the internal s_W signal is not yet updated and it outputs using the new iX and iY, but the old iW. This helps the clarity that I am loading the iW value and don't expect a significant output. Here is the updated test case:

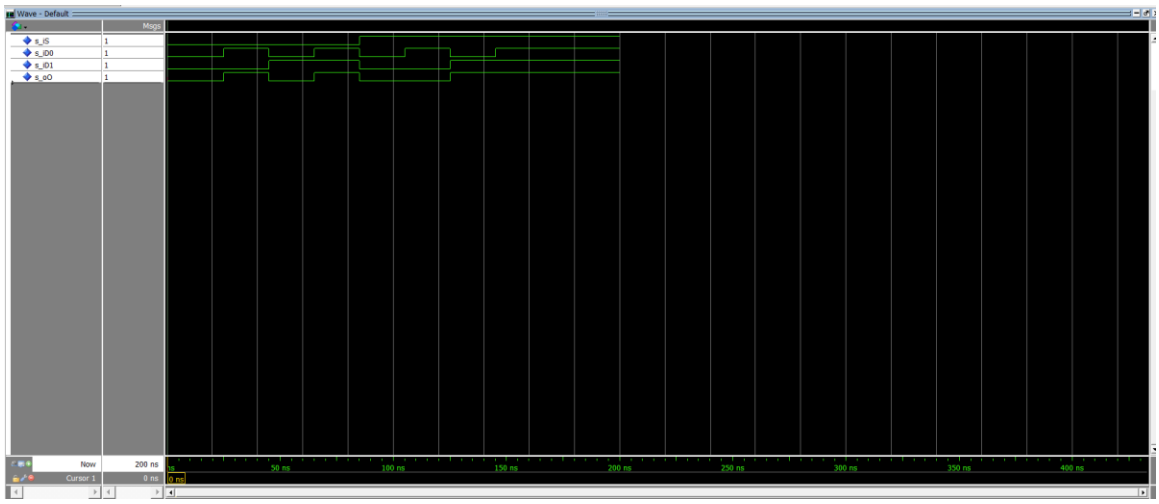
```
-- Test case 3:
-- Perform a MAC operation that results in oY being the same value as the day of my birth (21).
s_iX   <= x"00";  -- Not strictly necessary, but this makes the testcases easier to read
s_iW   <= x"03";  -- 3 in hexadecimal
s_iLdW <= '1';
s_iY   <= x"00";  -- Not strictly necessary, but this makes the testcases easier to read
wait for gCLK_HPER*2;
-- Expect: s_W internal signal to be 3 after positive edge of clock, oY will output 0 = 0*3+0 after 2 cycles.
s_iX   <= x"07";  -- 7 in hexadecimal
s_iW   <= x"00";  -- Not strictly necessary, but this makes the testcases easier to read
s_iLdW <= '0';    -- Make sure we don't continue to load.
s_iY   <= x"00";  -- 0 in hexadecimal
wait for gCLK_HPER*2;
wait for gCLK_HPER*2;
-- Expect: o_Y output signal to be 21 = 7*3+0 and o_X output signal to be 7 after two positive edge of clock.
```

[Part 3.3.a] Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 2:1 mux. Include this in your lab report.



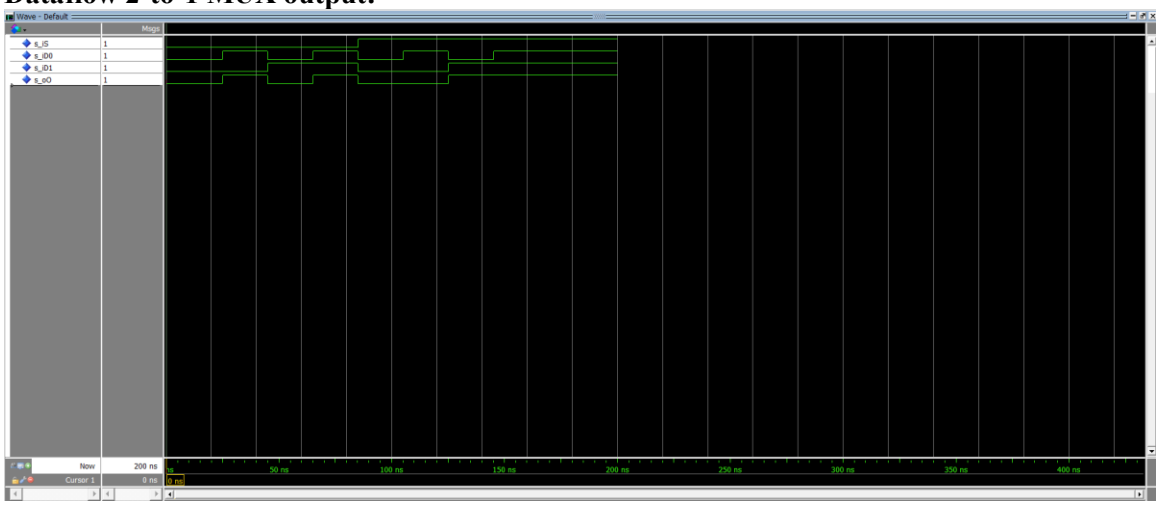
[Part 3.3.d] In your lab report, include a screenshot of the waveform. Make sure to label the screenshot with which module it is testing.

Structural 2-to-1 MUX output:

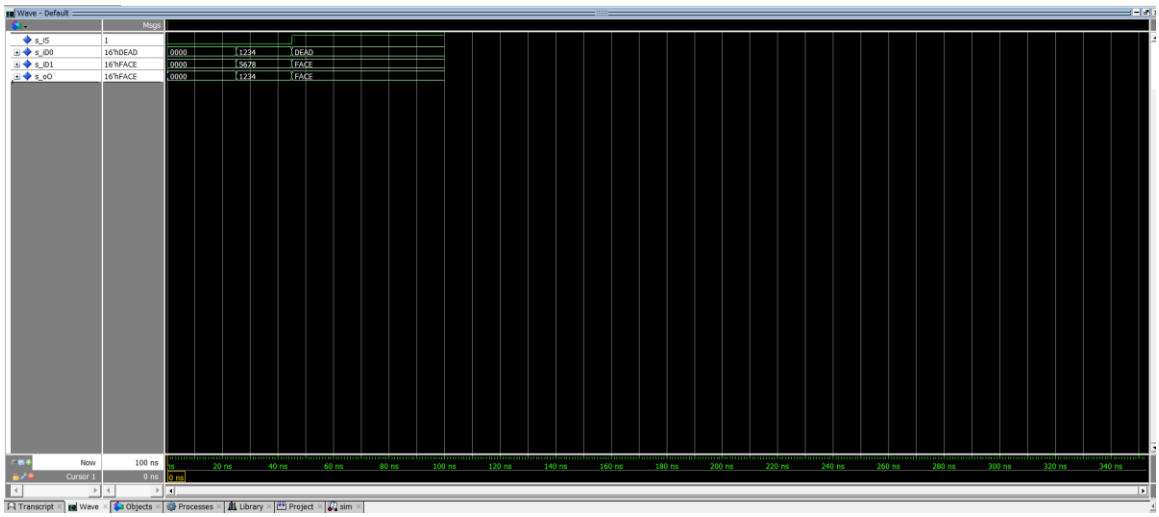


[Part 3.3.e] Again, in your lab report, include a labeled screenshot of the waveform showing the dataflow mux implementation working.

Dataflow 2-to-1 MUX output:

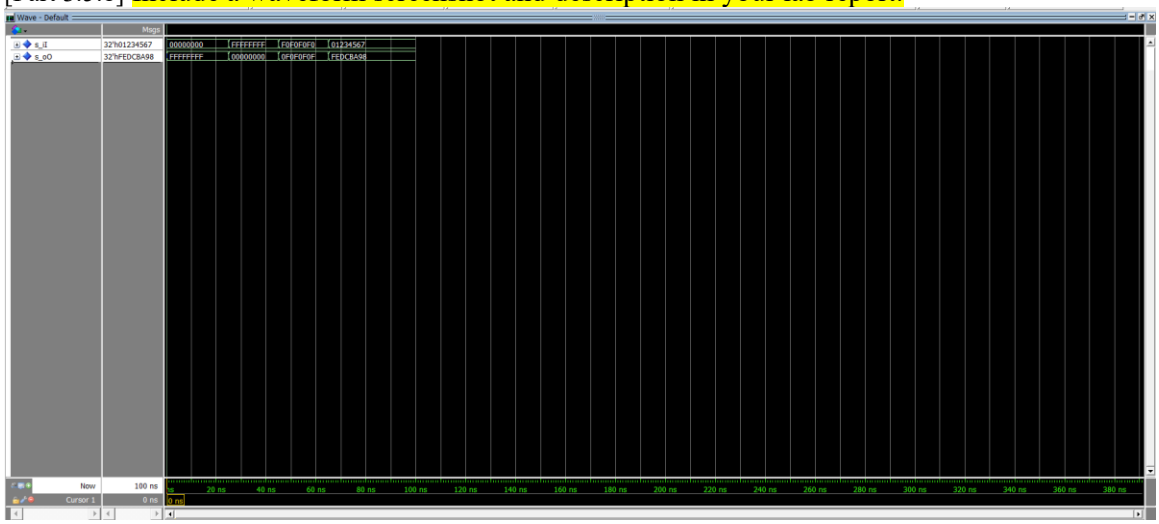


[Part 3.4] Include a waveform screenshot and corresponding description demonstrating it is working correctly.



As seen in the wave form, when s_iS is set to 0, s_oO matches the input of s_iD0. When s_iS is set to 1, s_oO matches the input of s_iD1. The first test case sets all inputs to 0, we expect s_oO to match the input of s_iD0 (set to 0x0000) and it does. The 2nd test case, we set s_iS to 0 and the other inputs to differing values. We expect s_oO to match the input of s_iD0 (0x1234) and it does. Finally, we set s_iS to 1 and the other two inputs to new differing values. We expect s_oO to match the input of s_iD1 (0xFACE) and it does.

[Part 3.5.b] Include a waveform screenshot and description in your lab report.



We know that one's compliment simply switches all of the bits of a binary number (from '0' to '1' or '1' to 0). So in the first test, when we input 0x00000000, we expect the output to be 0xFFFFFFFF and it is. In the 2nd test we input 0xFFFFFFFF and expect the output to be 0x00000000 and it is. In the 3rd test, we input 0xF0F0F0F0 and expect the output to be 0x0F0F0F0F and it is. In the final test, we input 0x01234567 and expect the output to be 0xFEDCBA98 and it is.

[Part 3.6.a] A full adder takes three single-bit inputs and produces two single-bit outputs – a sum and carry for the addition of the three input bits. Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 1-bit full adder. Include this in your report.

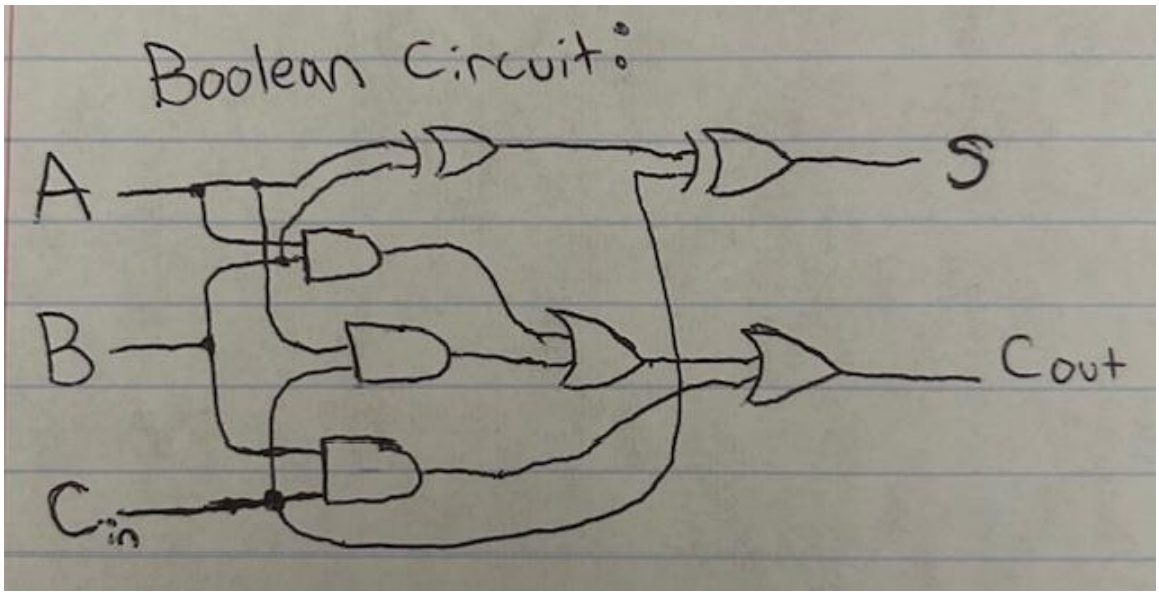
Truth Table:

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

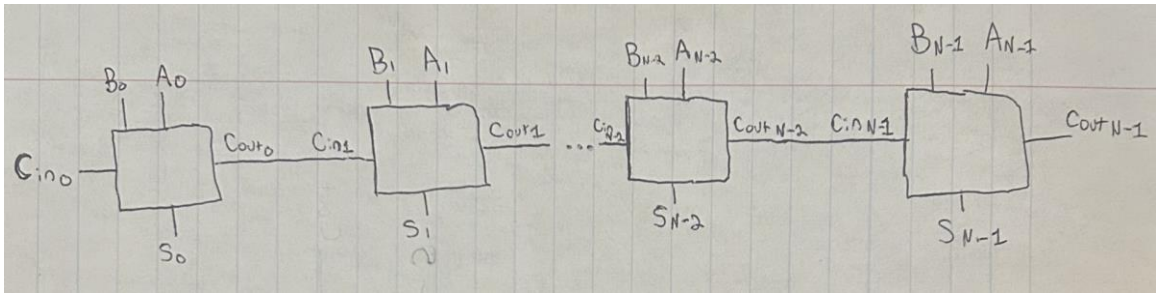
Boolean Equation:

$$(A \oplus B) \oplus C = S$$

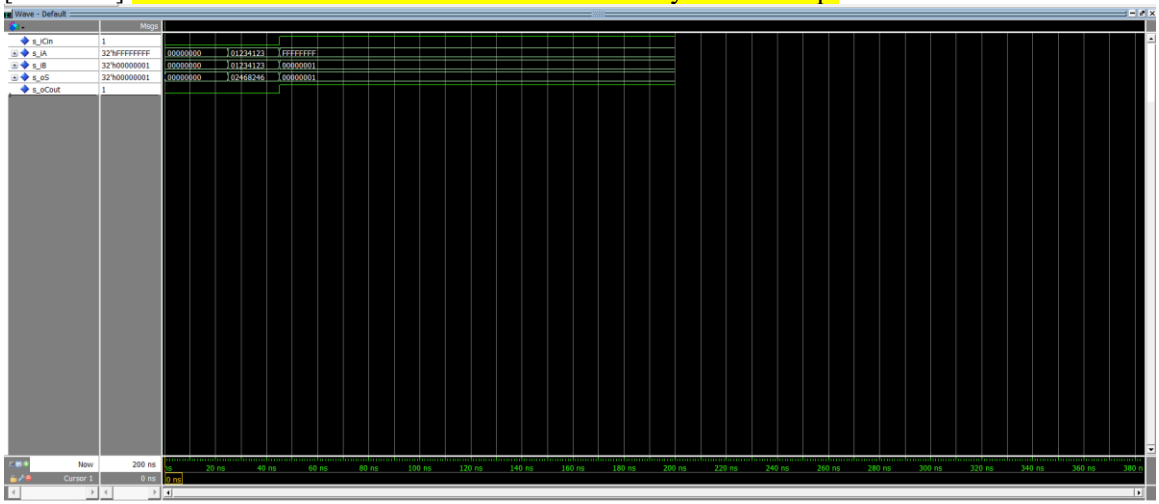
$$(AB) + (BC_{in}) + (AC_{in}) = C_{out}$$



[Part 3.6.c] Then draw a schematic of the intended design, including inputs and outputs and at least the 0, 1, N-2, and N-1 stages. Include this in your report.



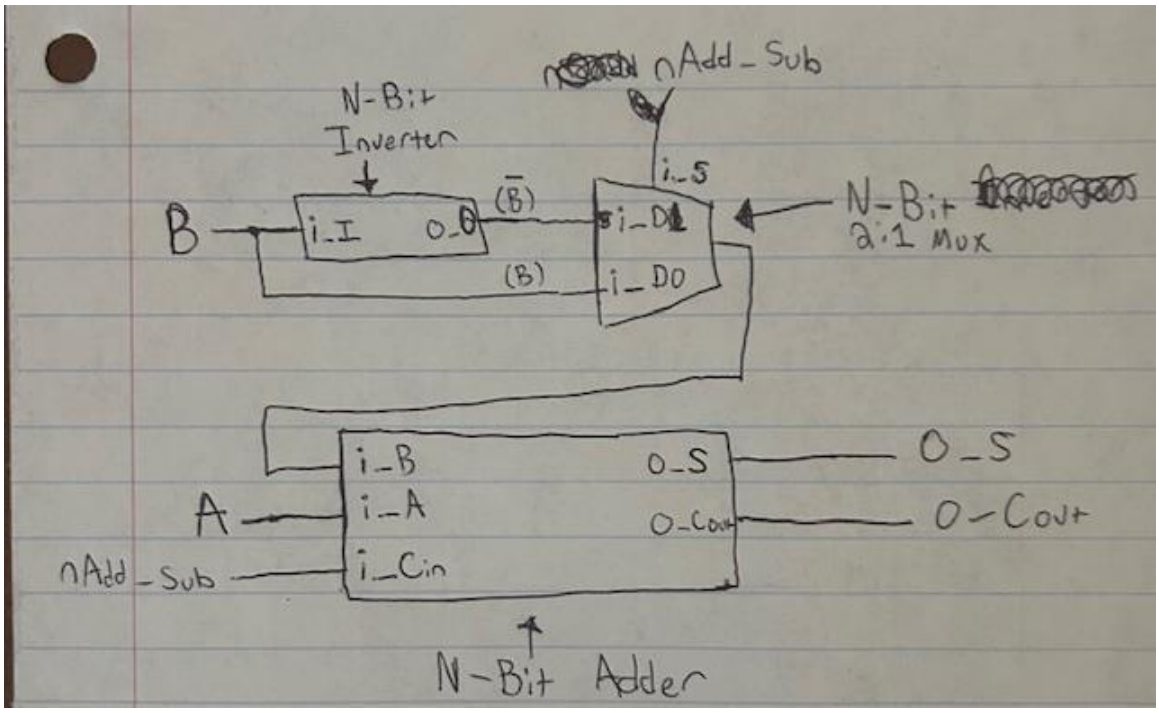
[Part 3.6.d] Include an annotated waveform screenshot in your write-up.



For the first test case, we set all of the inputs to 0 and expect the sum to be 0 with no carry (which is what we see). In the second test case, we do a simple addition of 0x01234123, 0x01234123 and expect the sum output to be 0x02468246 with no carry out (which is what we see). Finally, we test the carry-in and carry-out by adding

0xFFFFFFFF and 0x00000001 with the carry in set to 1. The output sum should be 0x00000001 with a carry out of 1 (again this is what we see).

[Part 3.7.a] Draw a schematic (don't use a schematic capture tool) showing how an N-bit adder/subtractor with control can be implemented using only the three main components designed in earlier parts of this lab (i.e., the N-bit inverter, N-bit 2:1 mux, and N-bit adder). How is the 'nAdd_Sub' bit used? Include this in your report.



'nAdd_Sub' is both the selector bit in the MUX and the Carry in bit in the adder (to make the 1's-compliment of B become the 2's-compliment of B if subtracting)

[Part 3.7.c] Provide multiple waveform screenshots in your write-up to confirm that this component is working correctly. What test-cases did you include and why?

Test case 1: to create an easy test case and see if anything funky is happening, I set all of the inputs to 0. Essentially creating 0+0=0. The expected results are the o_Cout and o_S

both equal 0 and that happened.

Test case 2: I wanted to test a basic addition case. Adding 0x01234123 to itself should output 0x02468246 which is what we see in the wave form. There is also no carry out which is what we expect.

Test case 3: I wanted to test a basic subtraction case. Subtracting 0x12121212 from 0x67678234 should output 0x55557022, and it did in the waveform. The carry out should be 1 (because no borrowing happened) and it is in the waveform.

Test case 4: I wanted to test an addition case with overflow. Adding 0x00000001 to 0xFFFFFFFF should result in o_S being 0x00000000 and the overflow should be 1. Both of which happened in the waveform

Test case 5: I wanted to test subtraction with a negative output (this implies borrowing as well). I subtracted 0x00000020 from 0x00000010. It should result in o_S being 0xFFFFFFF0 and o_Cout being '0' (because of borrowing). As seen in the waveform, this is what happend.

Test case 6: For my final test, I wanted to subtract equal numbers. So, I subtracted 0xABCDEF01 from itself. This gave me an output of o_S = 0x00000000 and o_Cout being 1. This is what is expected.