CprE 381, Computer Organization and Assembly Level Programming

Lab 1 Report

Student Name Colton Hazlett

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 1.c] Think of three more cases and record them in your lab report.

Case 3: Perform one MAC operation that have the partial sum output equal to your birthday

- Initialize a weight register by setting iLdW to 1 and load in the value 200 to iW. Then set the activation input, iX, to 509, and the partial sum input, iY, to 99.
- I expect that after two positive edges of the clock the activation output, oX, will be 200 and the partial sum output, oY, will be 101899=509*200+99

Case 4: Perform two MAC operations that will have the partial sum output equal to 200 by adding the partial sum output into the partial sum input on the second operation.

- Initialize a weight register by setting iLdW to 1 and then load a value of 2 into the weight, iW. Then set the activation input, iX, to 50 and the partial sum, iY, to 10.
- I expect after two positive edges of the clock the activation output, oX, will be 2 and the partial sum output, oY, will be 110=2*50+10.
- Then using the partial sum output, oY, set the partial sum input, iY, to 110 and the activation input, iX, to 45.
- I expect after two more positive edges of the clock the activation output, oX, will be 45 and the partial sum output will be 200=2*45+110.

Case 5: Perform two MAC operation that will have the partial sum output equal to 100 by adding the partial sum output into the partial sum input during the second operation. Also, you can not change the activation input from 5.

- Initialize a weight register by setting iLdW to 1 and then load a value of 2 into the weight, iW. Then set the activation input, iX, to 5 and the partial sum input, iY, 10.
- I expect after two positive edges of the clock the activation output, oX, is 5 and the partial sum output, oY, is 20=5*2+10.
- Then using the partial sum output, oY, set the partial sum input, iY, to 20 and by setting iLdW to 1 set the weight, iW to a value of 36.
- I expect after two positive edges of the clock the activation output, oX, is 5 and the partial sum output, oY, is 100=5*36+20.

[Part 1.e] For labels 9, 20, 32, and 33, 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

9.) It is a register that stores an integer, specifically it is storing the weight.
- Reg.vhd – lines 1-45
- TPU_MV_Element.vhd – lines 53-57, 92-100, 112-115

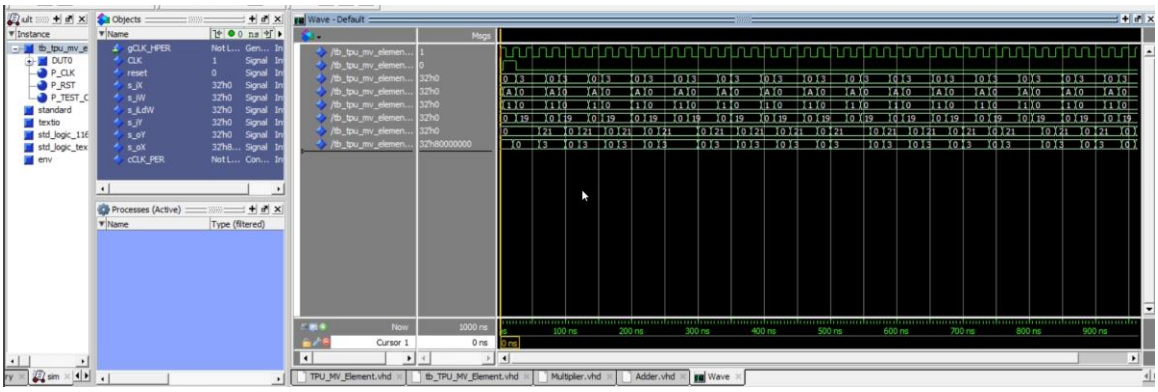
20.) iD is the input that the register is going to store on a positive edge clock.
- Reg.vhd – lines 29,37,40
- TPU_MV_Element.vhd – lines 55, 61, 94, 99, 114

32.) g_Add1 is a adder that takes two inputs iA and iB and then stores it in a register and outputs the sum to oC.
- Adder.vhd – lines 1 – 46
- TPU_MV_Element.vhd – lines 39-44, 117-121

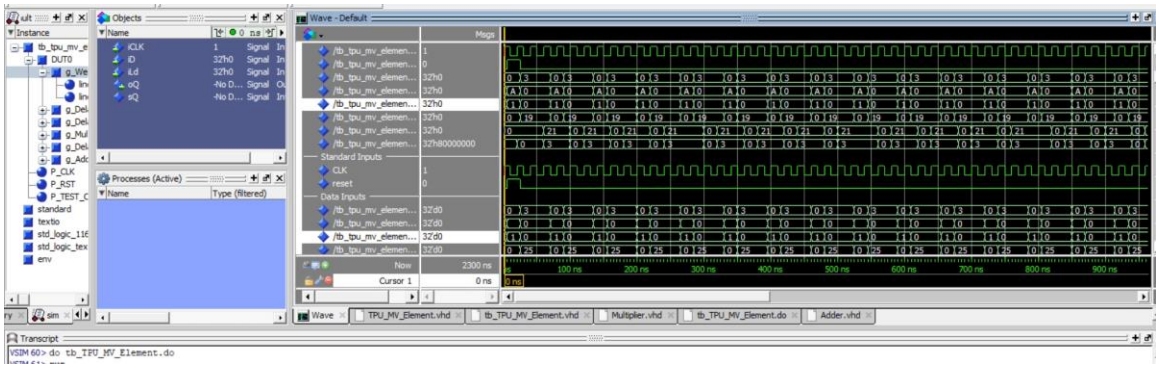
33.) Tensor Processing Unit it will take in a weight and store that until changed. Then it will multiple the weight times the value inputted at the activation input, iX, and add the value that is stored in the partial sum input, iY.
- TPU_MV_Element.vhd – lines 1-125
- tb_TPU_MV_Element.vhd – lines 34-42, 62 - 70

[Part 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 total that waveform looked about how I pictured that the element would function.



[Part 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'.



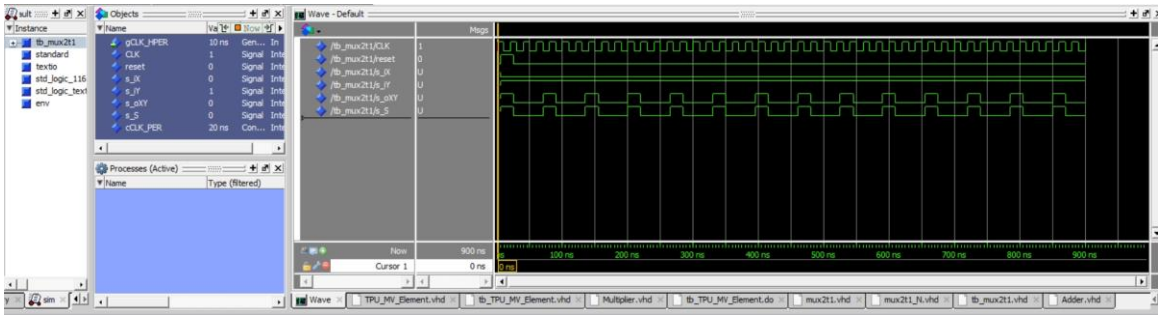
[Part 3.a] Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 2:1 mux. Include this in your lab report.

S_1	O
F	a
T	b

- $O = (\neg S_1 \wedge a) \vee (S_1 \wedge b)$
- $O = (S_1 \vee \neg a) \vee (\neg S_1 \vee \neg b)$

[Part 3.d] In your lab report, include a screenshot of the waveform. Make sure to label the screenshot with which module it is testing.

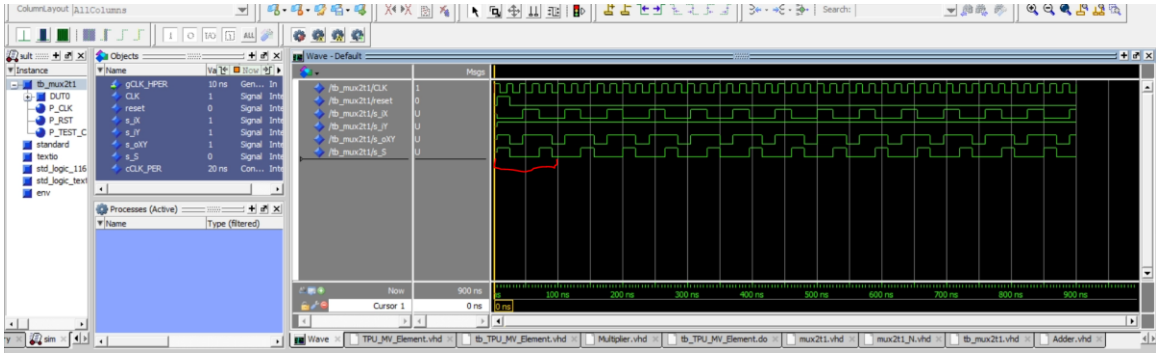
This is the structural wave…



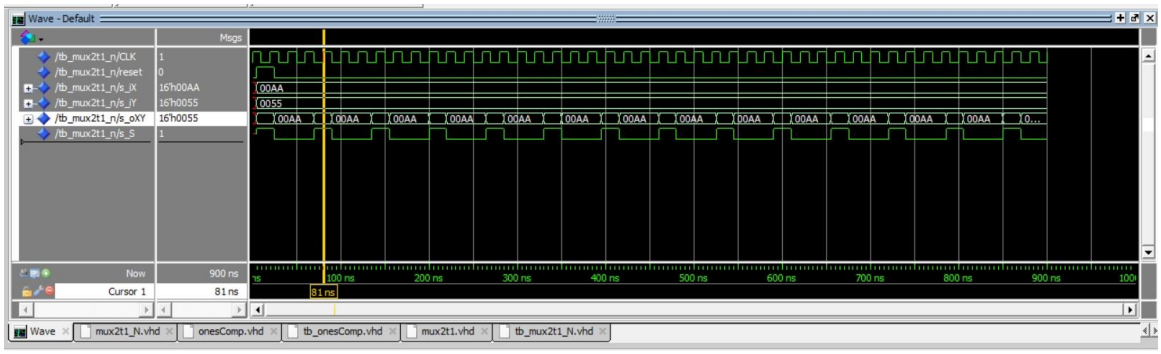
[Part 3.e] Again, in your lab report, include a labeled screenshot of the waveform showing the dataflow mux implementation working.

The following screenshot shos that my mux is working because in my test bench I set iX to output when i_S is 0 and set iY to output when i_S is 1/ Looking at the waves you can see the value that the select bit i_S selects is always outputted.

This is the dataflow wave…

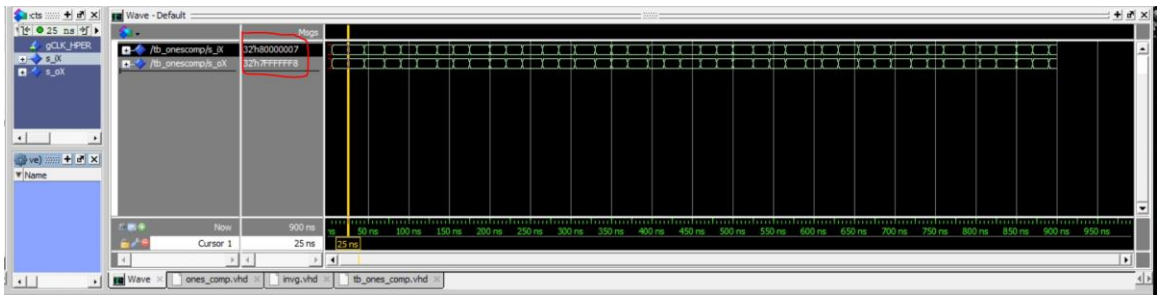


[Part 4] Include a waveform screenshot and corresponding description demonstrating it is working correctly.



I gave the test bench two test cases and that was just to switch between the two inputs by changing the select bit from 0 to 1. As you can see in the wave each time the s_S bit goes to 1 the output s_oXY changes from 0x00AA to 0x0055.

[Part 5.b] Include a waveform screenshot and description in your lab report.



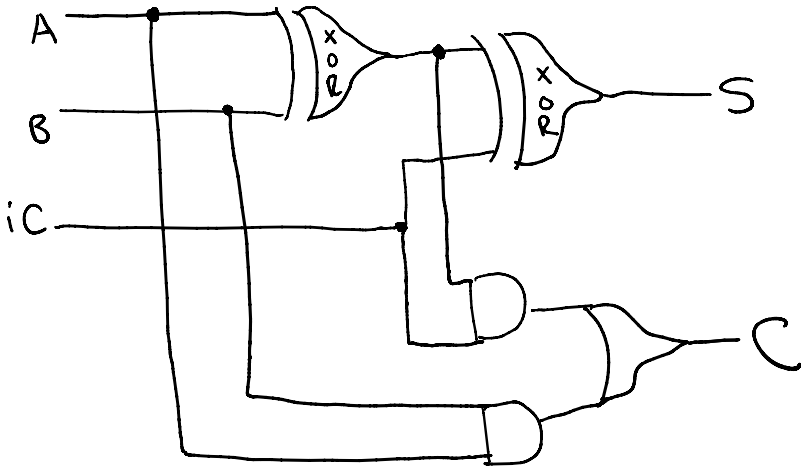
As you can see from the screen shot that is above the entry value s_iX was 32'h800000000007 and it switched to 32'h7FFFFFFFFF8.

[Part 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.

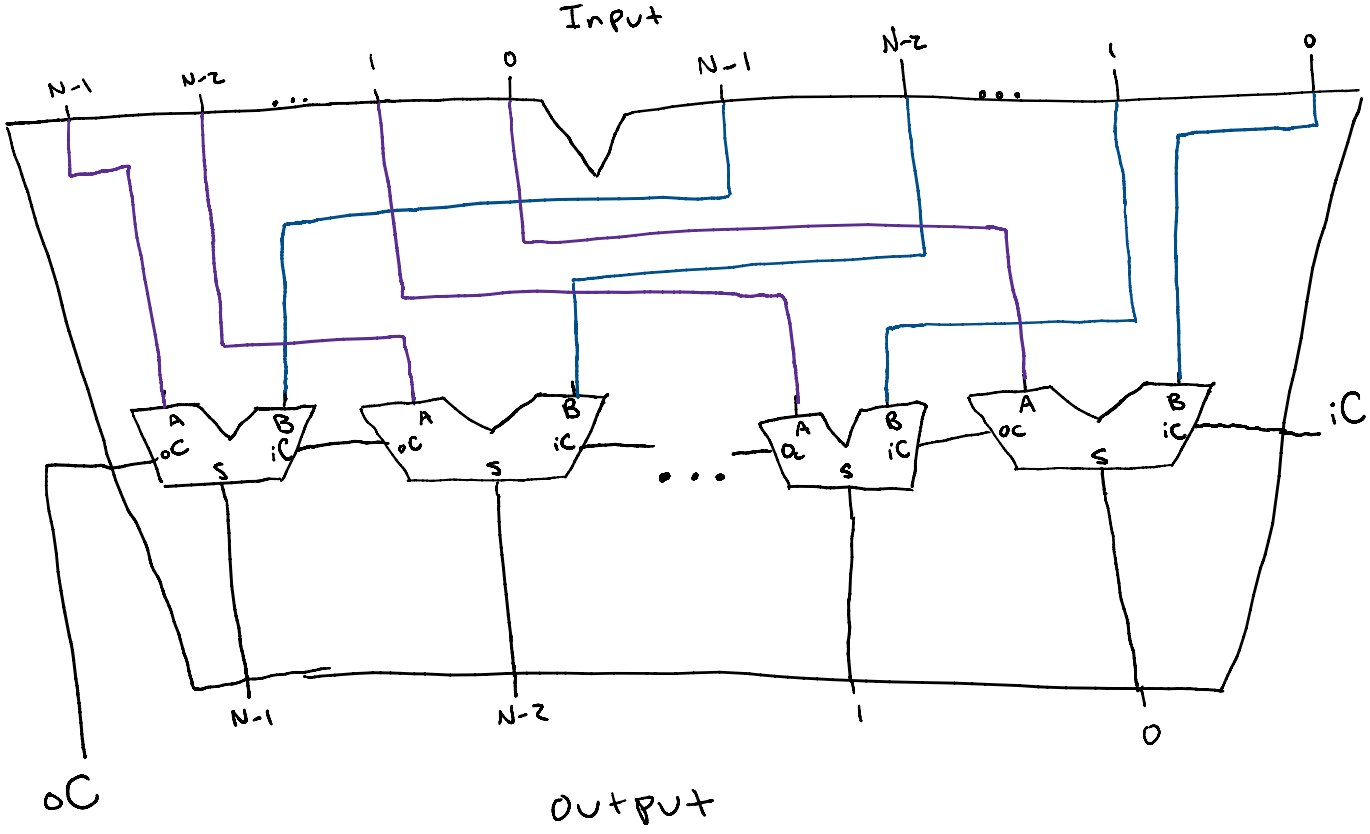
iC	A	B	S	oC
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \left(\underbrace{\left[(\neg A \vee B) \vee (A \wedge \neg B)\right]}_{\text{XOR}} \wedge \neg iC\right) \vee \left(\underbrace{\left[(A \wedge \neg B) \vee (A \wedge B)\right]}_{\text{XNOR}} \wedge iC\right)$$

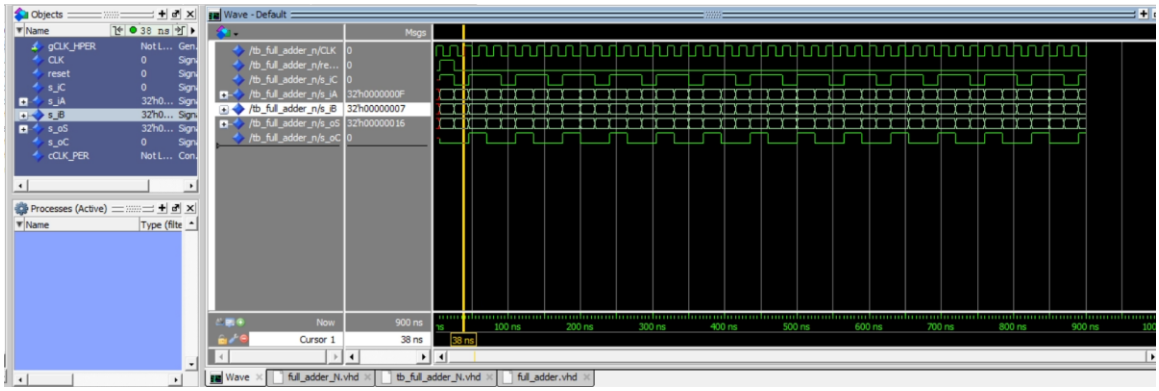
$$oC = \left[\neg iC \wedge (A \wedge B)\right] \vee \left[iC \wedge (A \vee B)\right]$$



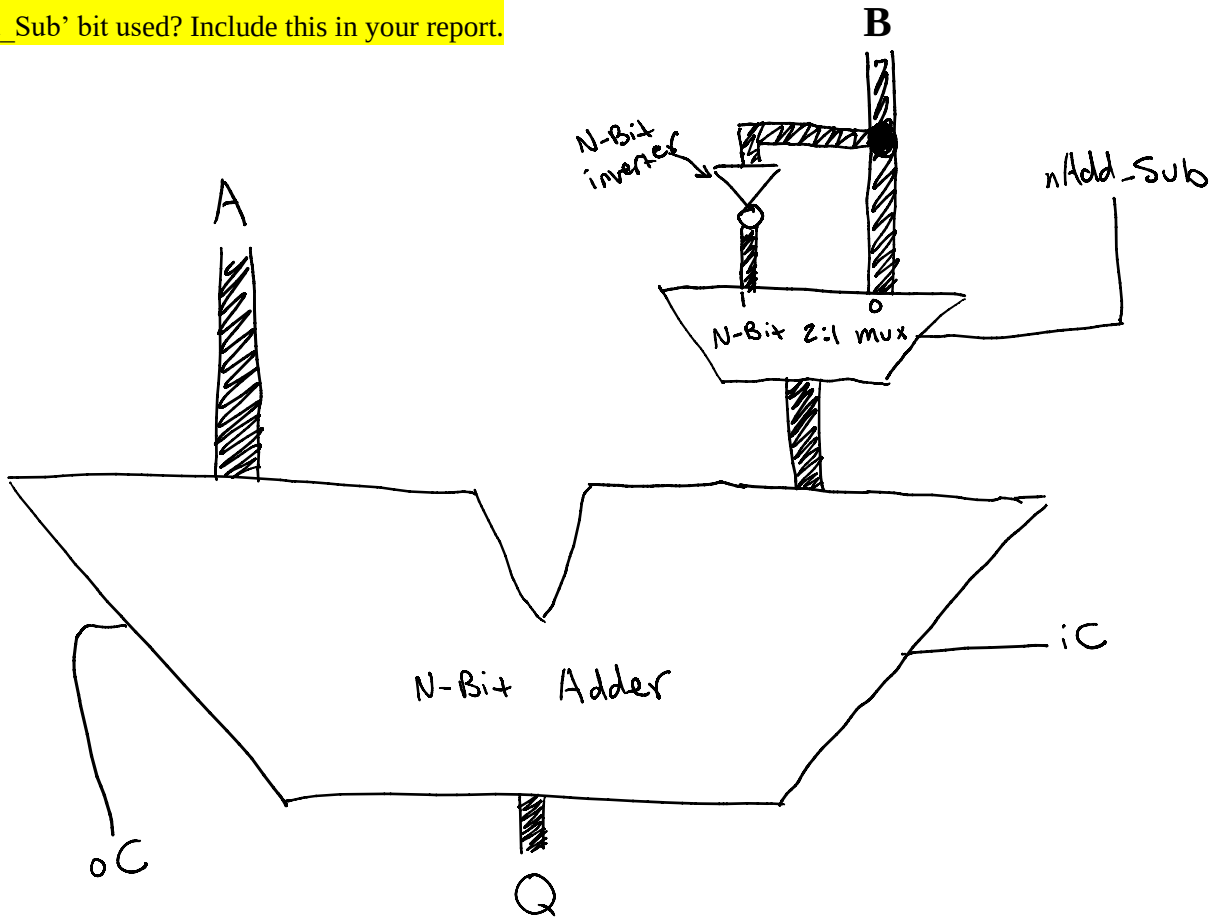
[Part 6.c] Then draw a schematic of the intended design, including inputs and outputs and at least the 0, 1, N-2, and N-1 stages. Include this in your report.



[Part 6.d] Include an annotated waveform screenshot in your write-up.

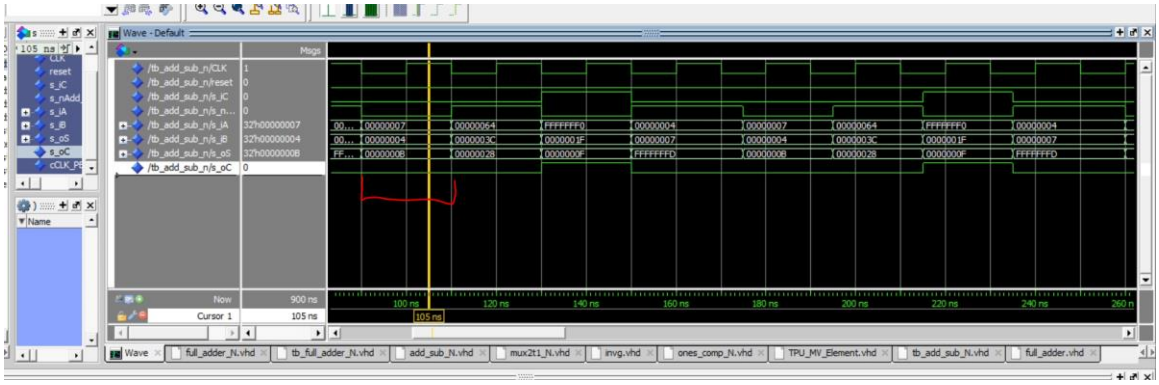


[Part 7.a] Draw a schematic (don't use a schematic capture tool) showing how an N-bit adder/subtractor with control can be implemented using only the three main components designed in earlier parts of this lab (i.e., the N-bit inverter, N-bit 2:1 mux, and N-bit adder). How is the 'nAdd_Sub' bit used? Include this in your report.



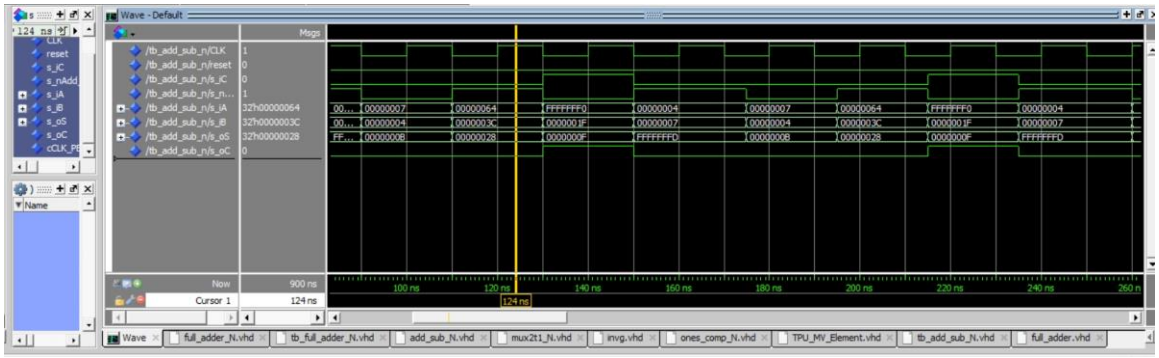
[Part 7.c] Provide multiple waveform screenshots in your write-up to confirm that this component is working correctly. What test-cases did you include and why?

Test Case 1:



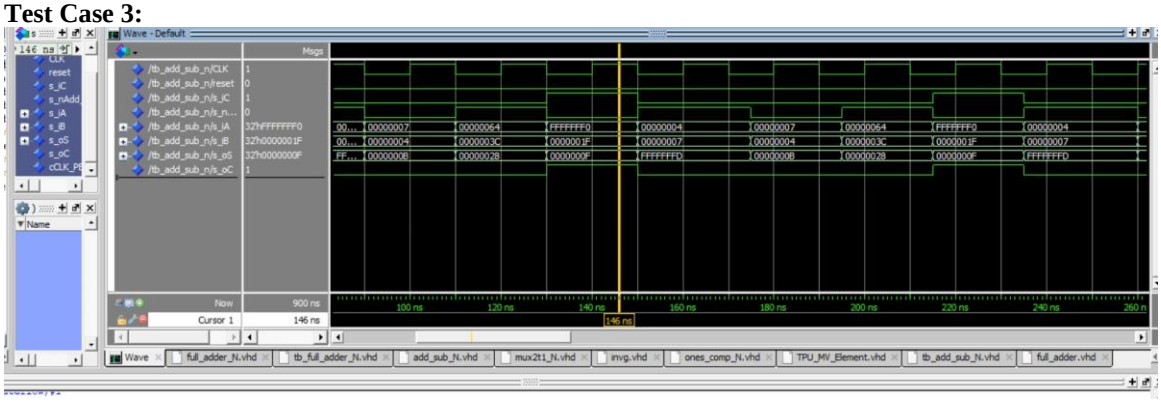
I created this test case to be a simple addition problem adding 0x7 + 0x4 and having o_S be 0xB with o_C equal to 0. I created this to show that I can add a simple problem.

Test Case 2:



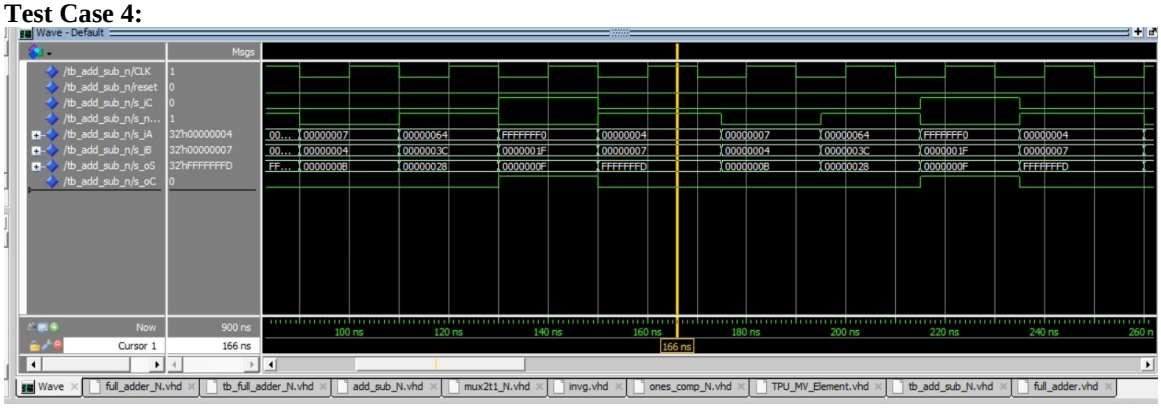
I created this test case to complete a simple subtraction by subtraction 100 (0x64) – 60 (0x3C) to have o_S equal to 40 (0x28). I created this to show that it can perform a simple subtraction as well.

Test Case 3:



I created this test to show that if you add two numbers that is larger than the supplied 32 bits than it will output that there is an overflow. I had the top 28 bits be set to 1 and then added 0x1F or 0b11111 so that there is an overflow. As you can see o_C is 1 and o_S went down to 0xF.

Test Case 4:



I created this one to show that when you subtract a number to a negative value it is precedded as an extremely large positive value because of how we are doing the ones complement subtraction.