CprE 381, Computer Organization and Assembly Level Programming

Lab 1 Report

Student Name Jonathan Hess

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: Resetting weight

Initialize the weight value register by setting iW to 7 and iLdW to 1 prior to a positive edge of the clock and resetting it after the positive edge. I then would set iW to 17 and re-enable iLdW to 1 for one clock cycle.

The value of s_w should be 17 after 1 clock cycle of setting iLdW.

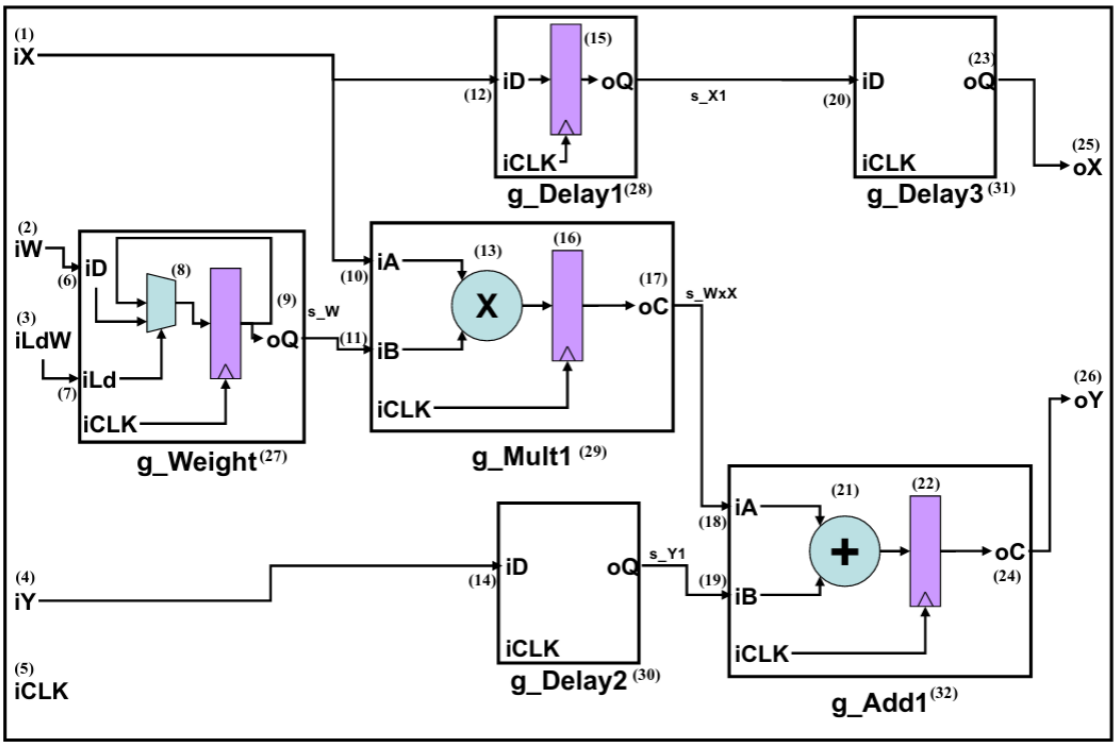
Case 4: Max Value

Set iX to FFFF and iW and iLdw = 1. iY = 0; after two clock cycles I would expect FFFF on both oY and oX.

Case 5:

Set iX to 27233 (0x6A6) iY to 2 and iLdW to 1 while iW is 3. After 2 cycles I would expect the output of oX to be 27233 and oY to be 81701 (0x13F25) or my birthday 8/17/01.

[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



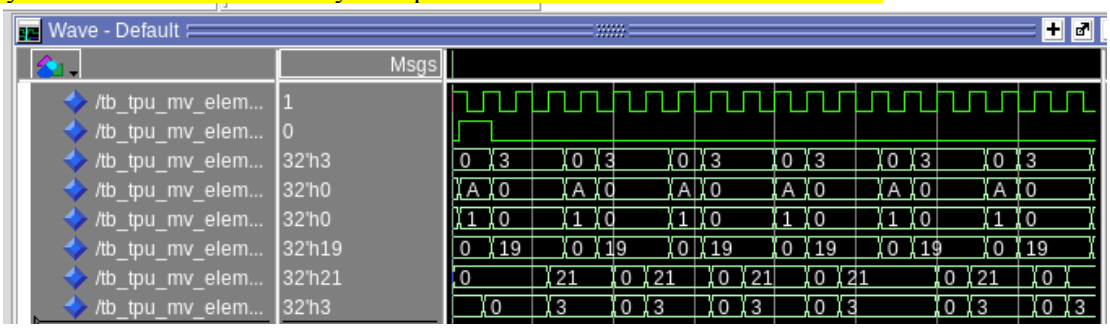
(9) is the oQ output of the register module, which is defined as an integer on Line 30 of Reg.vhd and set on Line 40. For this specific instance of the register module, the output oQ is connected as an input to the 0 condition of a 2-to-1 multiplexer (line 49 of TPU_MV_Element.vhd). It is also connected to g_Mult1 as iB as well as an adder in g_Add1 (line 49 of TPU_MV_Element.vhd). This is used to store the weight value loaded via iW and can be updated by changing iLdW.

(20) is the iD input of the g_Delay3 which is connected from the output of g_Delay1 by signal s_X1. (line 114 of TPU_MV_Element.vhd) This is a delayed signal of the input X.

(32) is the g_Add1 component that adds the s_WxX (the multiplied term) with iY. (line 117 of tpu_MV_Element.vhd)

(33) is the entire module itself, its file is TPU_MV_Element.vhd. It is also found in the testbench in file tb_TPU_MV_Element.vhd (line 34).

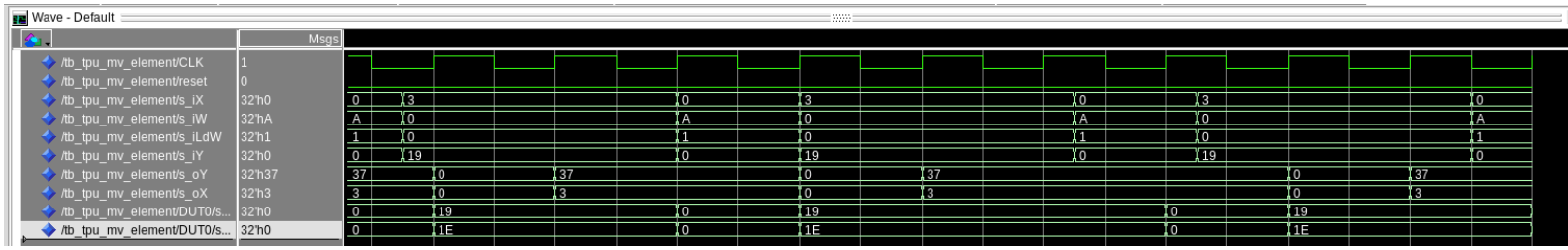
[Part 1.g.v] In your lab report, include a screenshot of the waveform. Describe, in plain English, any differences between what you expected and what the simulation showed.

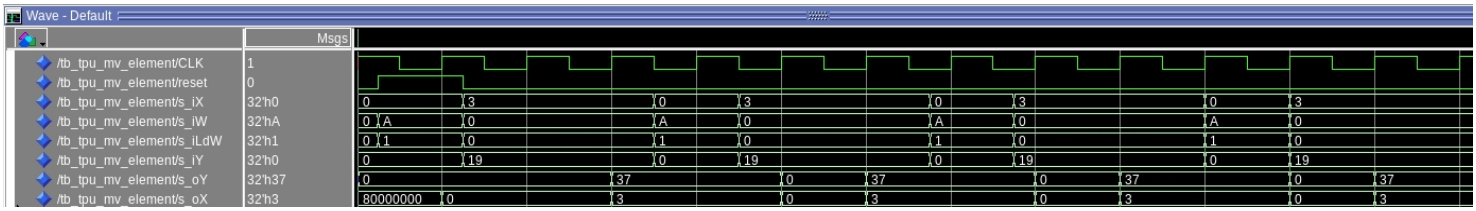


When looking at what the output should be we see that iX is 3 IW is 10 and IY is 25. This means that after the g_Multiply s_WxX should be 30 (which it is) but after the adder instead of 55 we get 33

[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'.

My waveform matched the expected output because 0*A + 0 = 0



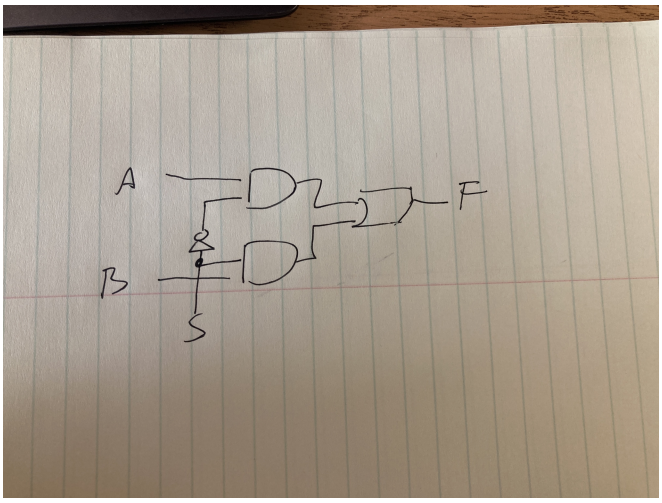


OY and OX are delayed by 2 cycles as would be expected(because the g_add1 also has delay)
The math works as well because

3*10(0xA) + 25(0x19) = 30+25 = 55 (0x37)
It stores the A value when the iLdw is set to 1 and ignores the 0 afterwards.

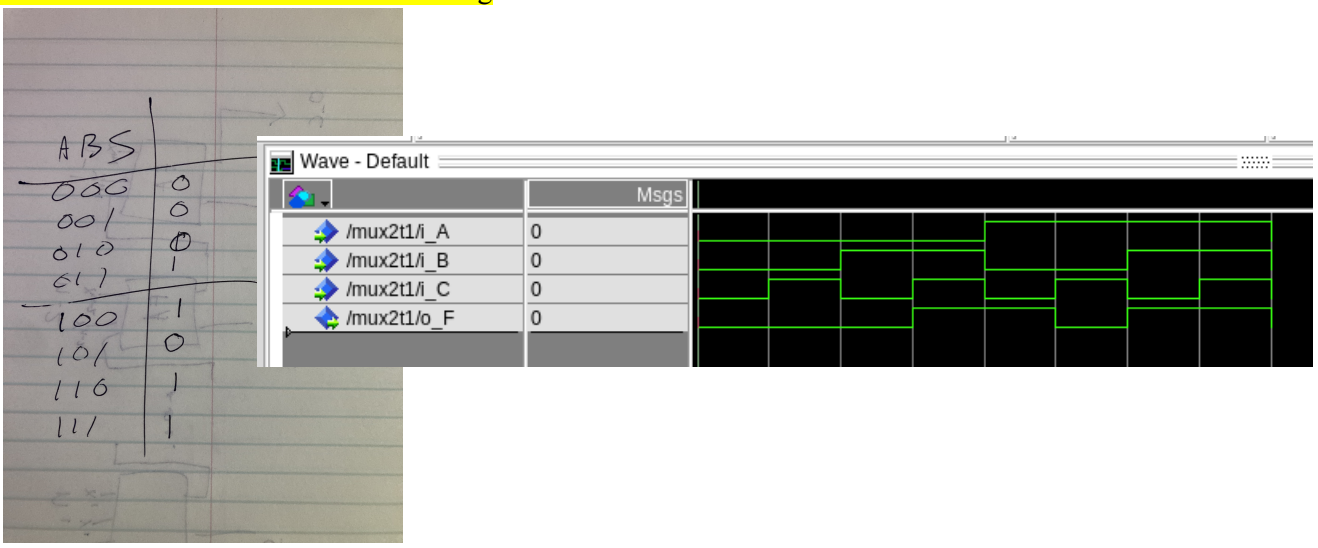
[Part 3.a] Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 2:1 mux. Include this in your lab report.

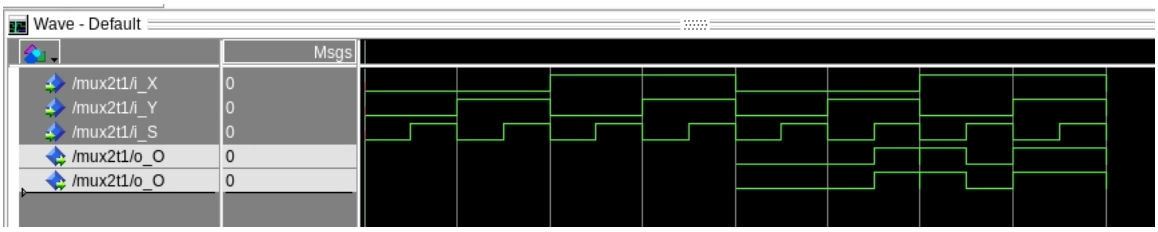
S	A	B	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



OR you could use a nand gate on the S input if you want to use only two input gates (no one input not)

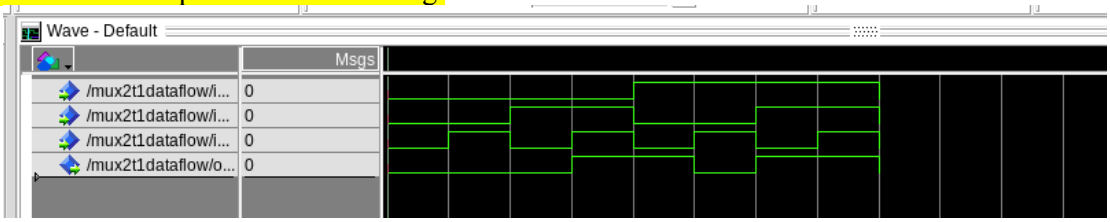
[Part 3.d] In your lab report, include a screenshot of the waveform. Make sure to label the screenshot with which module it is testing.





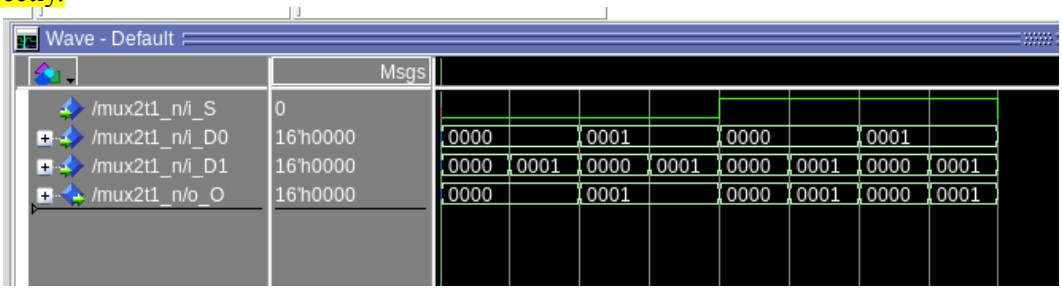
I originally used Dataflow for both so there may be errors with the NMUX because of different var names.

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



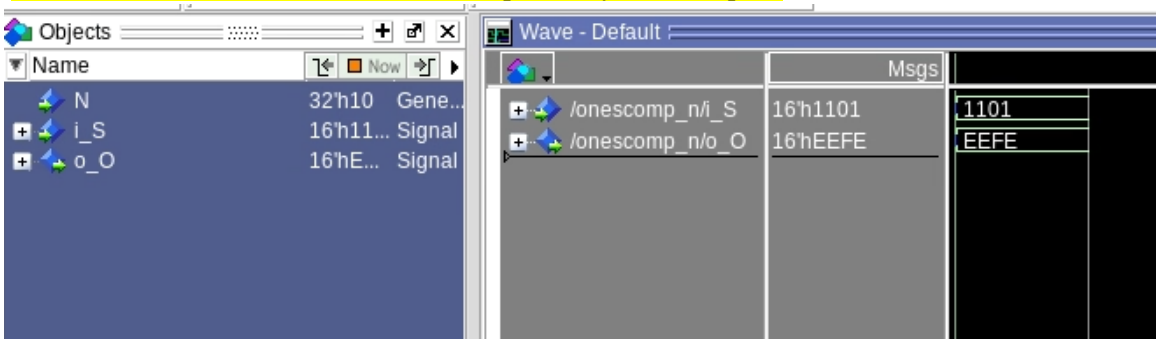
The Mux follows the TruthTable (if set to order ABS rather than SAB)

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



The mux works because it always outputs the value of the stream selected.

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

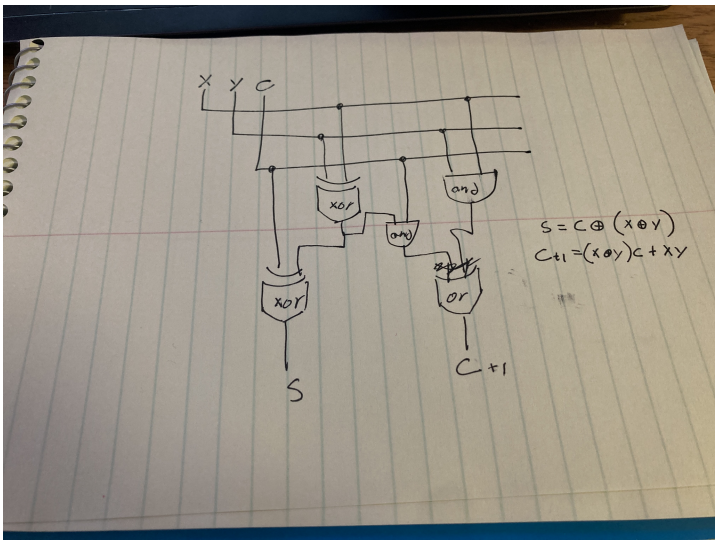


This inverses the inputs so 1101 becomes EEFE. I also tested with FFFF to get 0000.

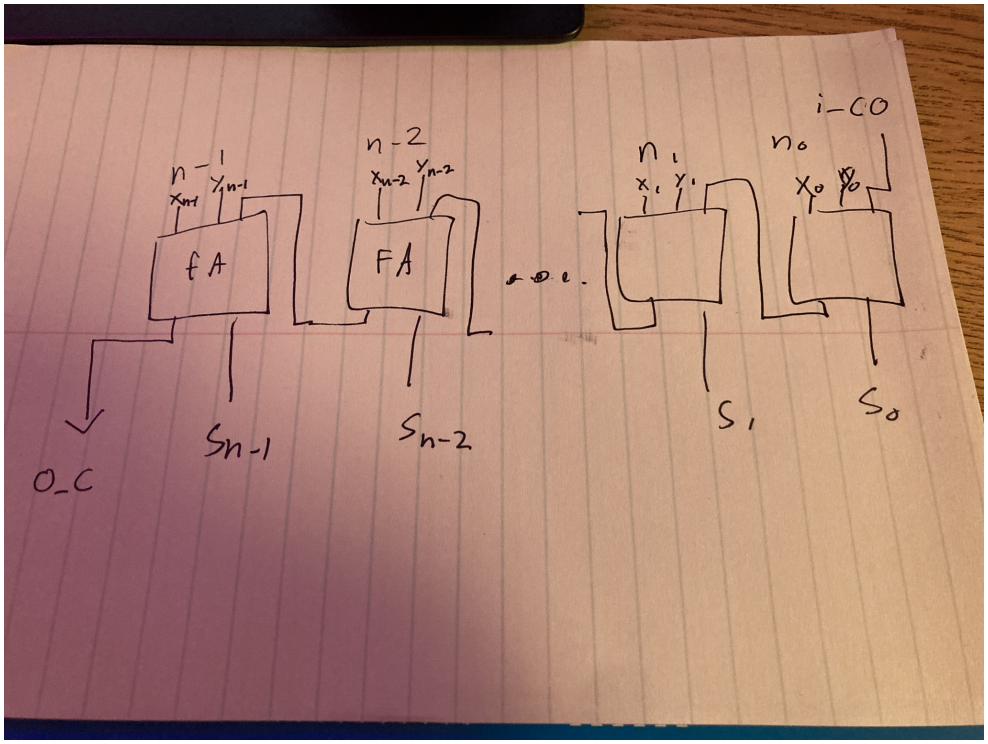
[Part 6.a] A full adder takes three single-bit inputs and produces two single-bit outputs – a sum and carry for the addition of the three input bits. Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 1-bit full adder. Include this in your report.

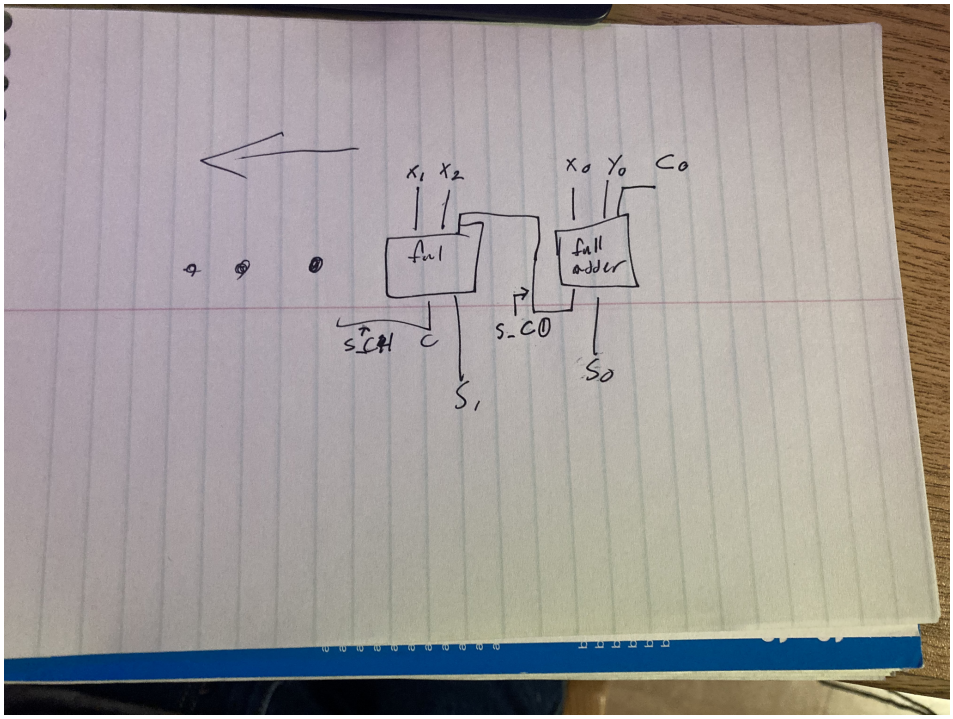
ONE BIT FULL ADDER

X	Y	C	C+1	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

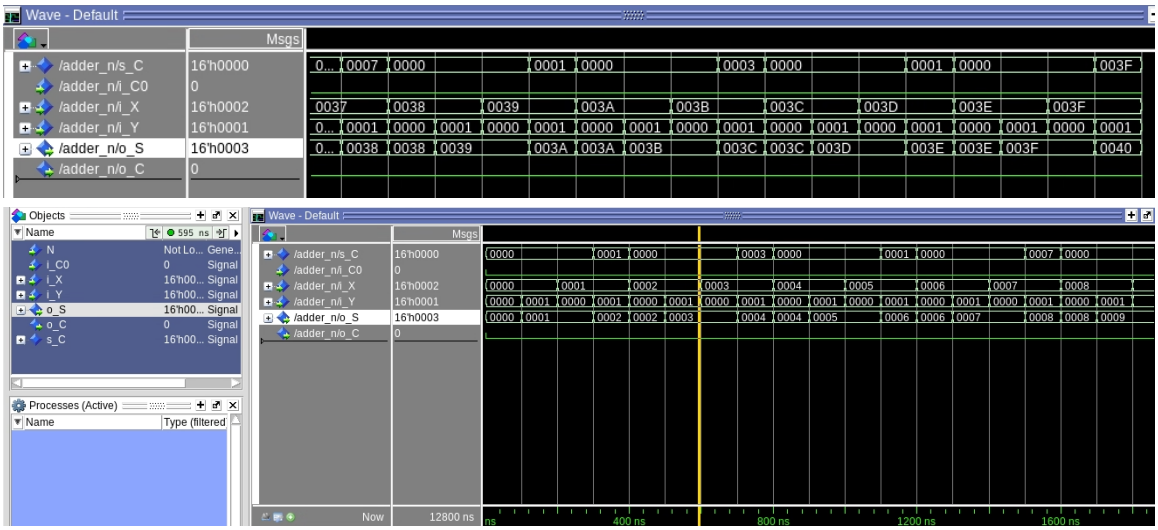


[Part 6.c] Then draw a schematic of the intended design, including inputs and outputs and at least the 0, 1, N-2, and N-1 stages. Include this in your report.



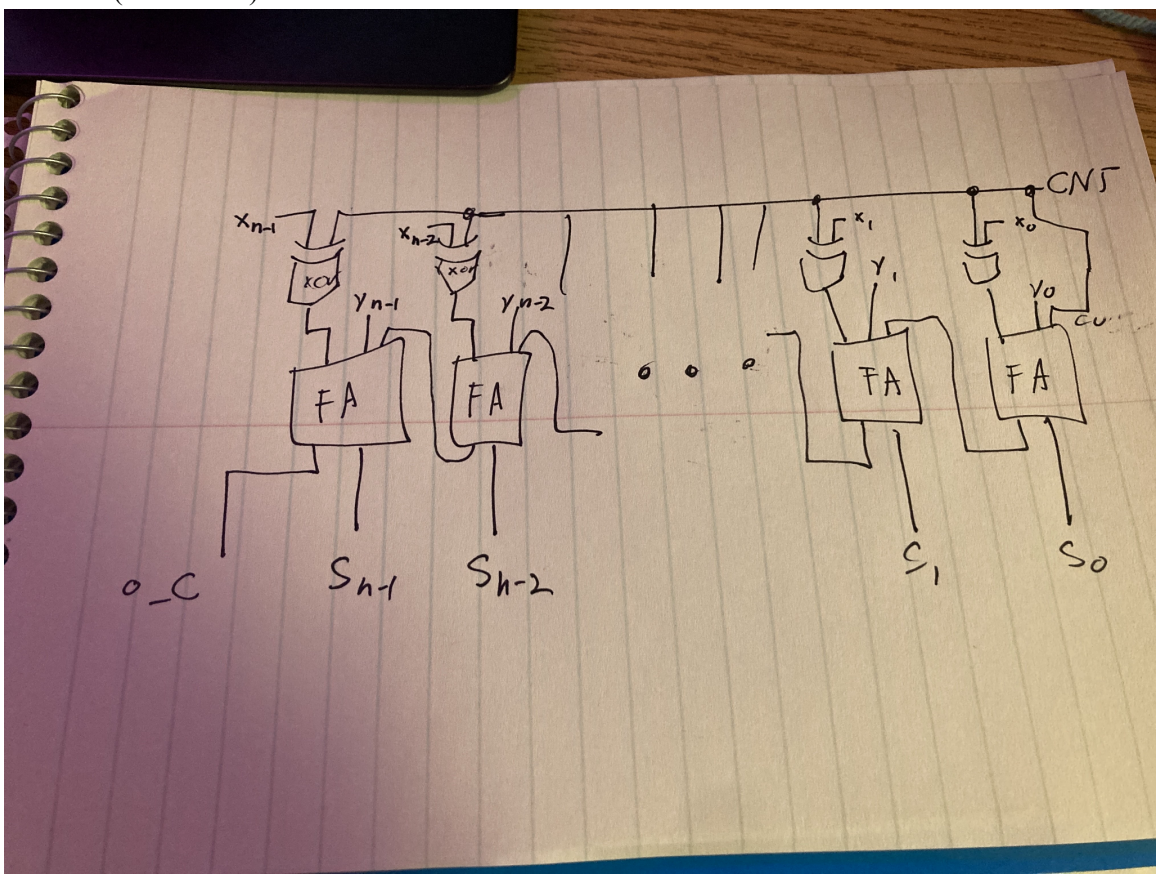


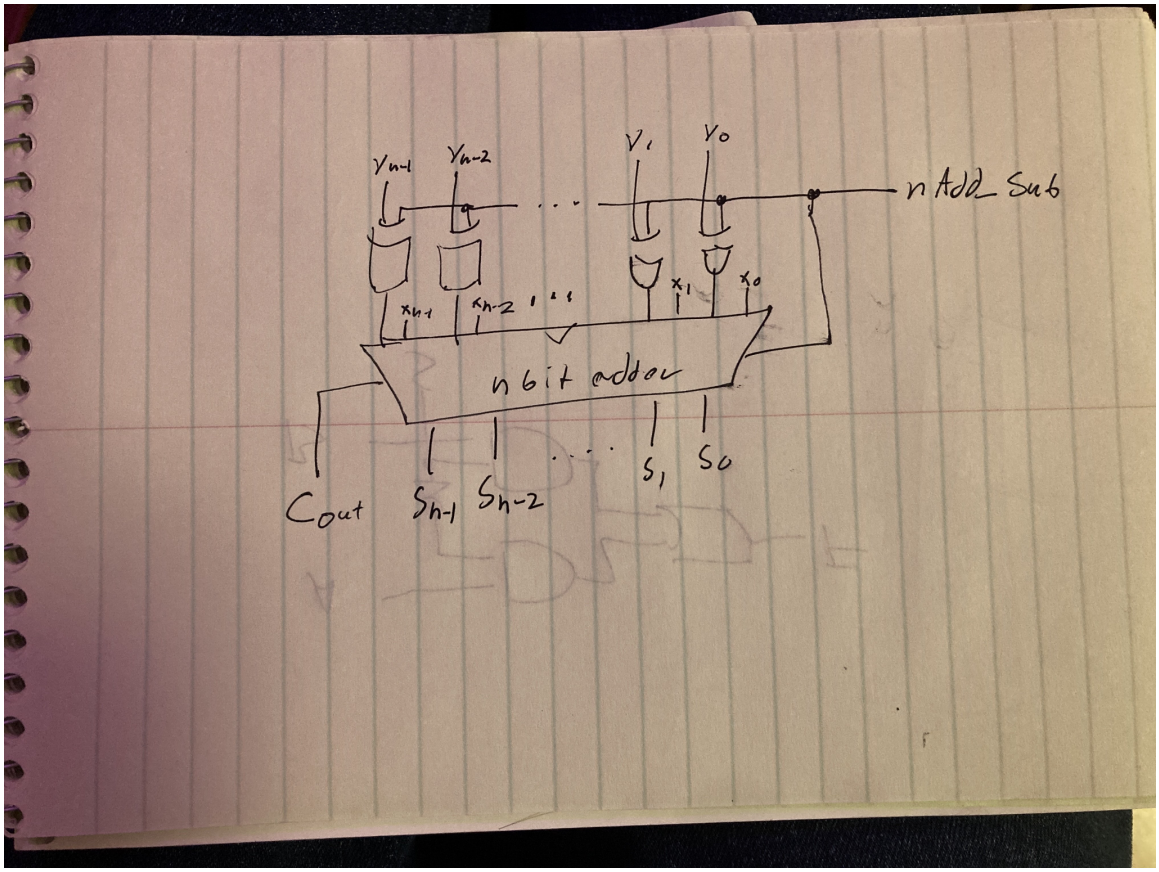
[Part 6.d] Include an annotated waveform screenshot in your write-up.



[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.

Using full adders (not correct)





The NAdd_Sub bit is used to 2s complement the secondary value (Y or B in this case) The xors will work as not gates when the NAdd_Sub is set to 1 and then it will add one by having a c0 of 1.

[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?

I tried both addition and subtraction limits by overflowing it. The final CN bit would show an overflow.