CprE 381, Computer Organization and Assembly Level Programming

Lab 1 Report

Student Name _____**Jacob Boicken**_____

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.

Perform one MAC operation with edge case values:
- Load the weight by assigning iW to 0 and iLdW to 1 and holding that until after the positive edge of the clock. Then, assigning iLdW to 0 to prevent the weight from changing. Then, set the activation value, iX, to 0 and the partial sum to 5.
- Then, after the two positive edges the activation output, oX, is 0 and the partial sum output is 5 (0*0+5).

Perform one MAC operation with edge case values:
- Load the weight by assigning iW to 12 and iLdW to 1 and holding that until after the positive edge of the clock. Then, assigning iLdW to 0 to prevent the weight from changing. Then, set the activation value, iX, to 4 and the partial sum to 0.
- Then, after the two positive edges the activation output, oX, is 4 and the partial sum output is 48 (12*4+0).

Perform one MAC operation that outputs my birthday:
- Load the weight by assigning iW to 6 and iLdW to 1 and holding that until after the positive edge of the clock. Then, assigning iLdW to 0 to prevent the weight from changing. Then, set the activation value, iX, to 3 and the partial sum to 1.
- Then, after the two positive edges the activation output, oX, is 3 and the partial sum output is 19 (6*3+1).

[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

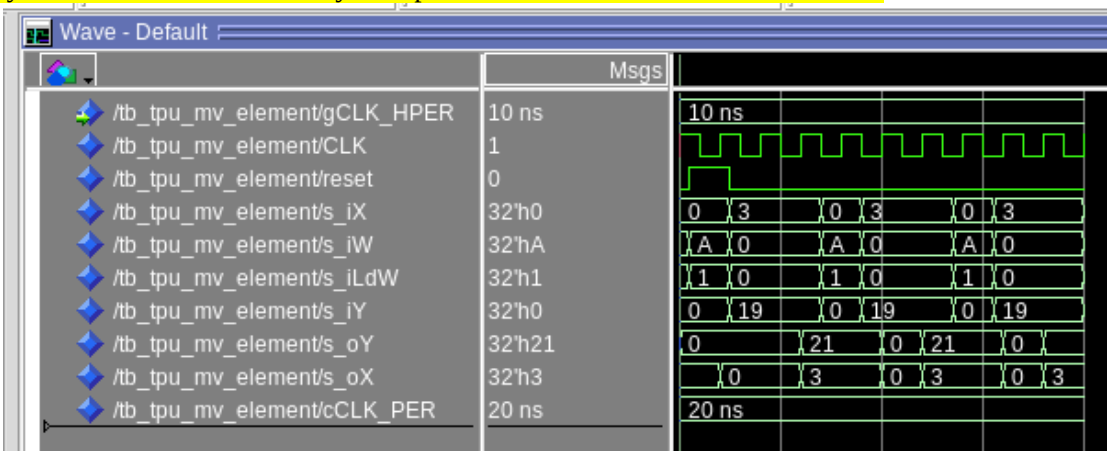
9. (oQ in g_Weight) – RegLd.vhd line 31 and TPU_MV_Element.vhd line 82.
This value outputs the stored value of iW, if iLdW is 1 or oQ, itself, if iLdW is 0, after a rising edge to in iCLK. This output is sent to g_Mult1 to be multiplied to iX.

20. (iD in g_Delay3) – Reg.vhd line 29 and TPU_MV_Element.vhd line 112.
This inputs the value stored in g_Delay1 to then be stored in its g_Delay3, after a rising edge in iCLK. This delays the output of oX to be in sync with oY.

32. (g_Add1) - TPU_MV_Element.vhd line 117 and Adder.vhd line 26.
This component adds the value of iW * iX received from g_Mult1 with the value of iY stored in g_Delay2. This value is stored after a rising edge in iCLK and the result is outputted to oY.

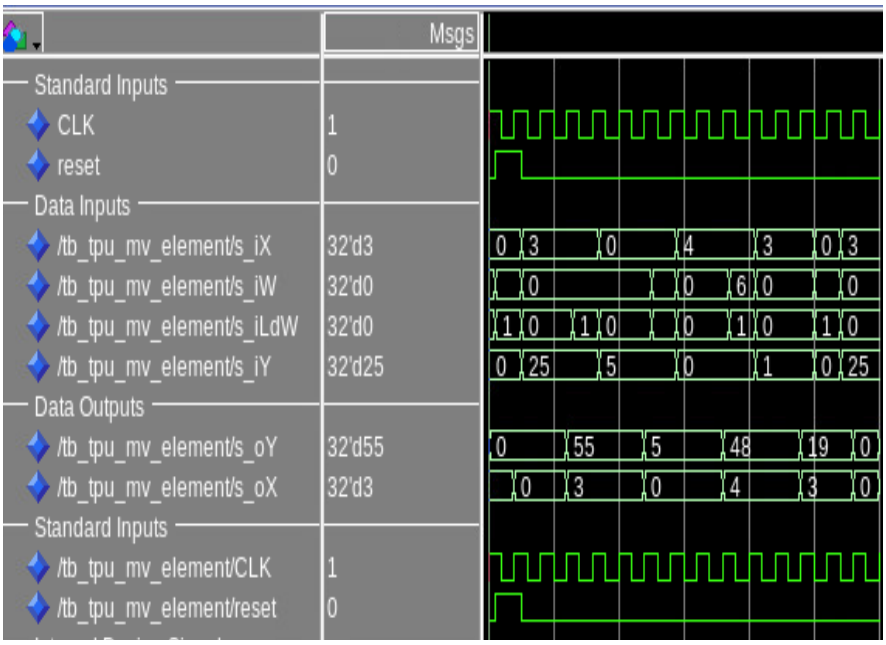
33. (TPU_MV_Element) - TPU_MV_Element.vhd line 23 and tb_TPU_MV_Element.vhd line 62.
This is the complete component of the entire TPU. It takes in the iX, IY, iW, and iLdW inputs and outputs the oX and oY values.

[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.



So the outputs are updating after two clock cycles (rising edges). However, oY is not the value of iX * iW + iY. But oX is correct being updated to the value of iX after two clock cycles.
For instance, iX is 3(3), iW is not set but the previously held value is A(10), and iY is 19(25), but oY is 21(33) and should be 37(55).

[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'.



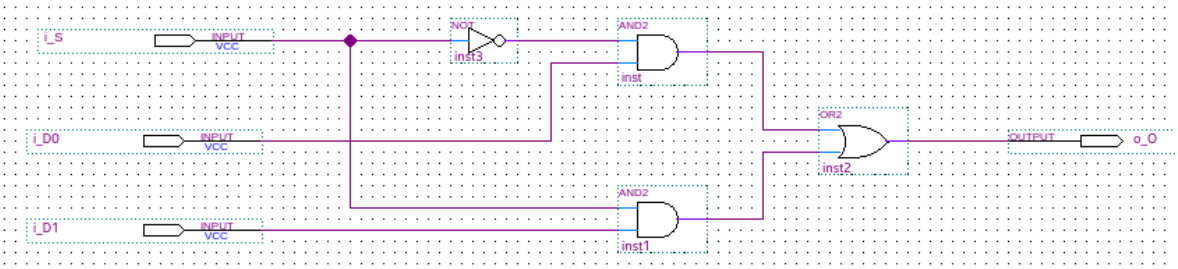
At the start, iX is 0, iW is A, iLdW is 1, and iY is 0. The first cycle loads in iW, iX, and iY, which we can tell since s_W, s_X1, and s_Y1 all reflect that. For next 2 cycles, iX is 3, iY is 19, iW is 0 and iLdW is 0, so only iX and iY are loaded into s_X1 and s_Y1, respectively. While s_W is still held as A. As well, s_WxX is updated to 1E from zero during the second cycle, which is correct for A * 3 or 10 * 3 = 30 in decimal (s_W * s_X1). During the third cycle, oX is updated from 0 to 3 reflecting the input of iX, which is intended. Then, oY is updated correctly to 37, which is s_WxX + s_Y1 or 1E + 19 or 30 + 25 (decimal). These outputs are then held for 2 cycles, since it takes 2 to multiple iW and iX then add that with iY. (And oX being synced to that as well.)

(I only described the preset tests. I also have one that only test iY adding 5, only iX and iW multipying to 48, and my birthday on the 19th with 3*6 +1.)

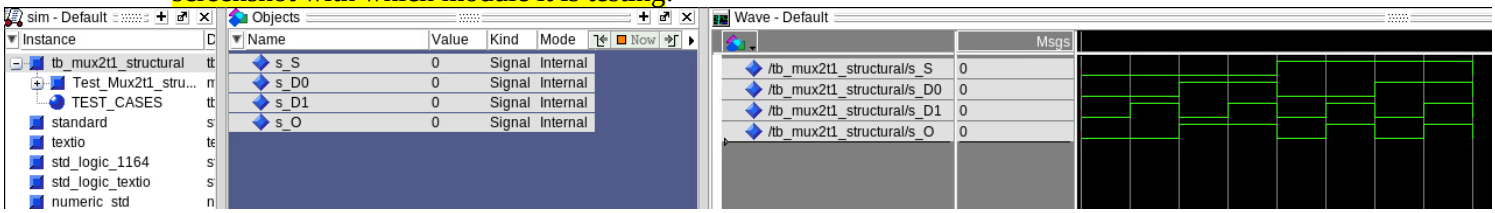
[Part 3.a] Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 2:1 mux. Include this in your lab report.

i_S	i_D0	i_D1	o_O
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

o_O = ~i_S * i_D0 + i_S * i_D1

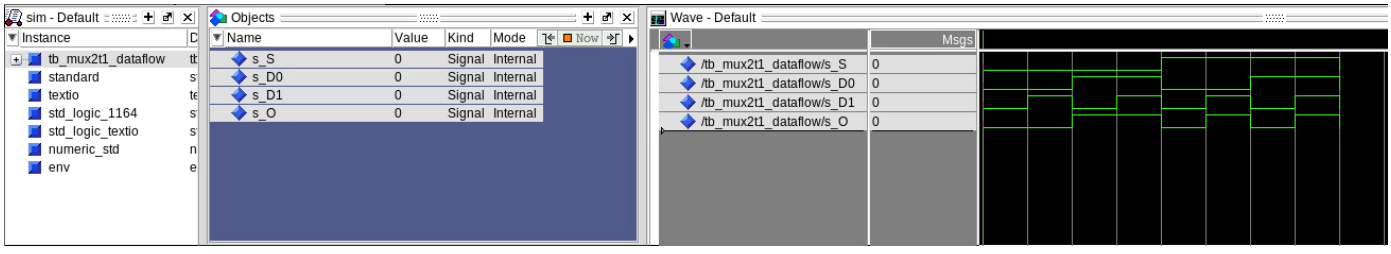


[Part 3.d] In your lab report, include a screenshot of the waveform. Make sure to label the screenshot with which module it is testing.



This is the structural one as you can see by the right hand side with sub-components. It goes through all 8 possible inputs of the 2:1 mux.

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

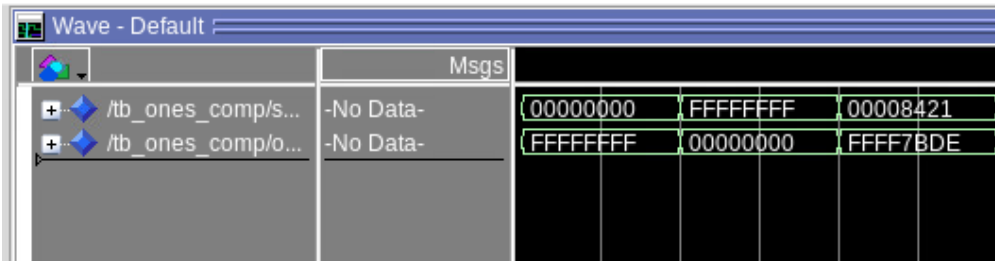


Dataflow both have the same output matching the table I made with the same test cases.

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

I set all the bits to opposite values for D0 and D1. I then flipped i_S from 0 to 1 to test if all bits will flip from D0's value of 0 to D1's value of 1. Since I tested the underlining multiplexer to work in all cases, I only need to know if multiplexer is working for each bit. So I think this would be sufficient to show that.

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



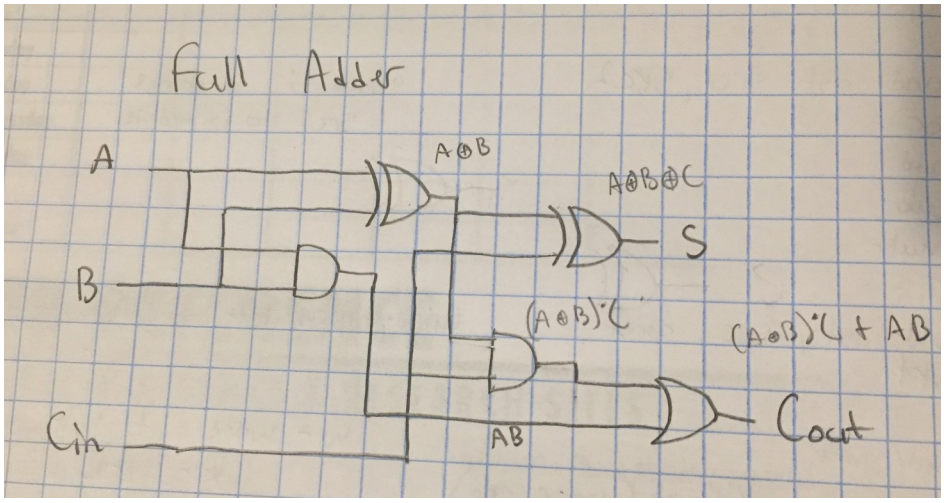
I tested if each bit flips completely as the generic I used should flip each bit, using the working invg component, from the inputted on in the N-length array. The first two test that each bit flips from 0 to 1 and 1 to 0. The last was to test each individual bit alone in the set of 4. These all flip to the correct values, 1 to E, 2 to D, 4 to B and 8 to 7.

[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.

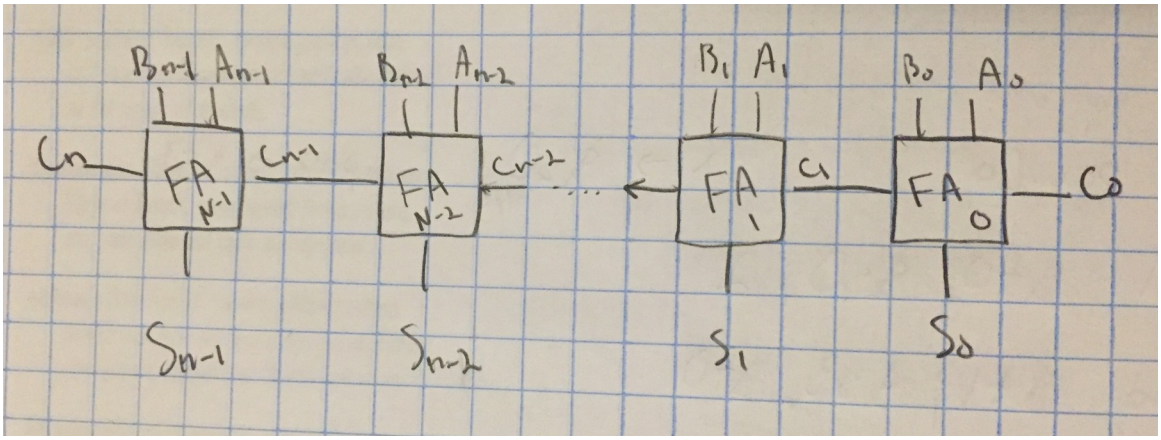
A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Cout = AB + Cin (A xor B)

S = A xor B xor Cin



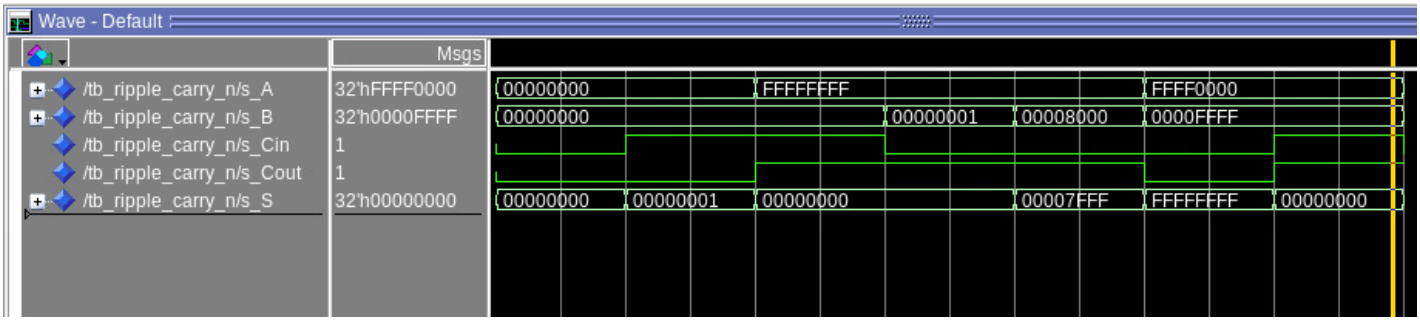
[Part 6.c] Then draw a schematic of the intended design, including inputs and outputs and at least the 0, 1, N-2, and N-1 stages. Include this in your report.



Some expected cases of this would be:

A	B	C0	S	Cn
0xFFFFFFFF...	0x00000000...	1	0x00000000...	1
0xFFFFFFFF...	0x00000000...1	0	0x00000000...	1
0x00000000...	0x00000000...	0	0x00000000...	0

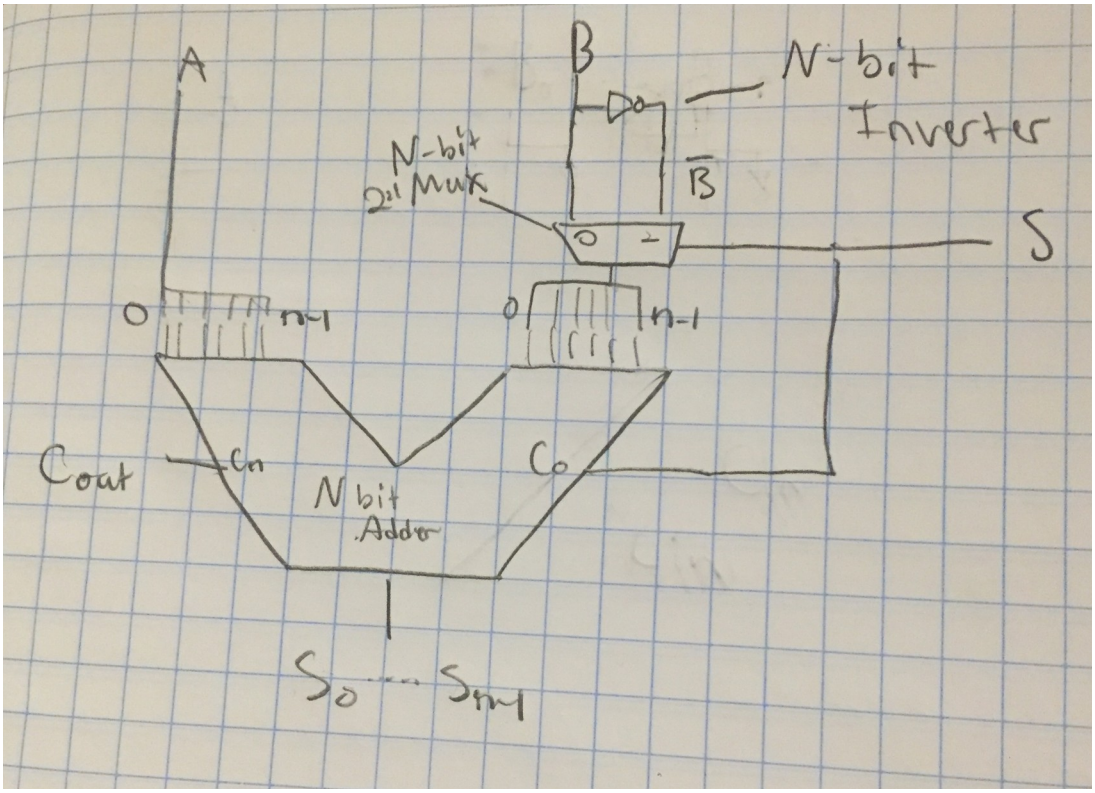
[Part 6.d] Include an annotated waveform screenshot in your write-up.



Waveform for my ripple-carry adder. I tested it by checking how the bits overflow in third, fourth, fifth, and seventh cases, if Cin works properly in the first and second cases, and normal additions to ensure the sixth case.

[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.

The Selector bit for adding or subtracting is used as the select bit for the N 2:1 mux where S being one puts the value of ~B to be added and S being zero B is added. As well, the S bit is entered into as C0 of the N bit adder to complete two's complement input of inverting + 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 did 14 cases to test the adder/subtractor. I did 7 different A B input pairs in both adding and subtracting modes. (First 4) I tested with all zeros to be added/subtracted with to test if the system flips the bits correctly checking the Cout being high in subtracting mode. (5-8) I also tested adding/subtracting by 1 to check the adder overflowing correctly but positively and negatively. (11-14) I also tested adding/subtracting by all ones to further test overflows and getting the back one when subtracting zero all one. (9-10) I use the number 0x00008000 to add/sub against all ones to test if it overflows for only the remain bits above the 16th bit and subtracts only that 16th bit.