CprE 381, Computer Organization and Assembly Level Programming

Lab 1 Report

Student Name __________Austin Beinder______________________________

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.

1. Test Case 1
   a. iW = 5, ildw = 1, clk=lo, hi, lo, hi, ix = 2, iy = 4
   b. I expect that after 3 positive edges of the clk, oX will be 10 and oY will be 14
2. Test Case 2 – Doesn't change
   a. iW = 5, ildw = 1, clk=lo, hi, lo, hi, lo, hi ix = 2, iy = 4
   b. I expect that after 4 positive edges of clk, oX will be 10 and oY will be 14
3. Test Case 3 - ildw shouldn't matter if it isn't 1 on a clock pulse
   a. iW = 5, ildw = pulsing, but 0 on each positive clk pulse, clk=lo, hi, lo, hi, lo, hi ix = 2, iy = 4
   b. I expect no change in the output if ildw is pulsed 10 times but is 0 during each positive clock pulse and if iW is changed each clock cycle by adding 1.
4. Test Case 4 – Birthday
   a. Iw = 1, ildw = 1, clk= lo, hi, lo, hi, lo, hi, ix = 72900, iy = 99
   b. I expect oY to output oY = 72999. I was born July 29, 1999

[Part 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

Label 1: TPU_MV_ELEMENT -> TPU_MV_Element.vhd line 23
In this instance, TPU_MV_Element is the entity name as defined in TPU_MV_Element.vhd on line 23 which holds the whole architecture of the unit.

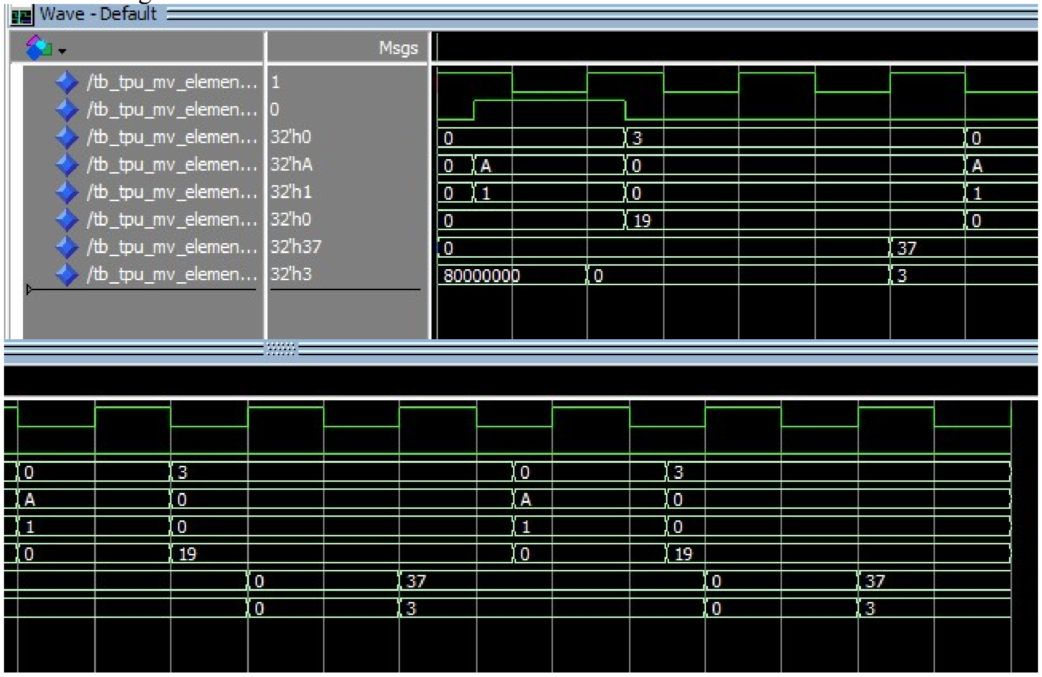
Label 7: g_Add1 -> TPU_MV_Element.vhd line 117
g_Add1 is the name of an element of the type Adder that is used within level 2 of the architecture and is defined on line 117 of TPU_MV_Element.vhd.

Label 22: oQ in g_Weight -> RegLd.vhd line 31
oQ is the name of the output port of the g_weight element that is a RegLD entity and is defined at RegLD.vhd line 31 internally, and its interaction in the top level design is defined in TPU_MV_Element.vhd on line 86.

Label 28: id in g_Delay3 -> Reg.vhd line 29
id is the name of the input of the g_Delay3 entity and is defined as a Reg.vhd input. It is defined where it interfaces with the design in TPU_MV_Element.vhd line 114 and is defined internally at Reg.vhd line 29.

[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.

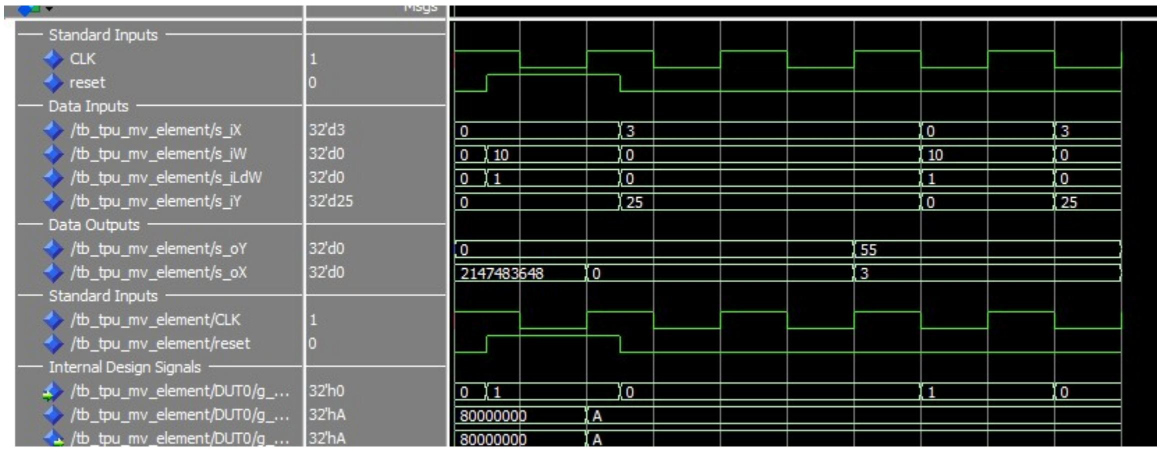
Non Working Waveform



```
Wave - Default
                         Msgs
/tb_tpu_mv_elemen...     1
/tb_tpu_mv_elemen...     0
/tb_tpu_mv_elemen...     32'h0
/tb_tpu_mv_elemen...     32'hA
/tb_tpu_mv_elemen...     32'h1
/tb_tpu_mv_elemen...     32'h0
/tb_tpu_mv_elemen...     32'h37
/tb_tpu_mv_elemen...     32'h3
```

Some of the differences between what I expected and what I observered are that I expected the output to be 55 and not 37. I also expected iY to become 25 and not 19.

[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'.



```
                                          Msgs
Standard Inputs
    CLK                                   1
    reset                                 0
Data Inputs
    /tb_tpu_mv_element/s_iX               32'd3
    /tb_tpu_mv_element/s_iW               32'd0
    /tb_tpu_mv_element/s_iLdW             32'd0
    /tb_tpu_mv_element/s_iY               32'd25
Data Outputs
    /tb_tpu_mv_element/s_oY               32'd0
    /tb_tpu_mv_element/s_oX               32'd0
Standard Inputs
    /tb_tpu_mv_element/CLK                1
    /tb_tpu_mv_element/reset              0
Internal Design Signals
    /tb_tpu_mv_element/DUT0/g_...         32'h0
    /tb_tpu_mv_element/DUT0/g_...         32'hA
    /tb_tpu_mv_element/DUT0/g_...         32'hA
```

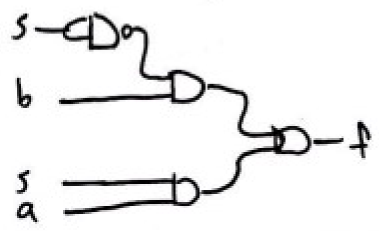
After correcting the problem, the output matches the expected behavior. As you can see oY becomes 55, and oX becomes 3, which is how it should be functioning.

[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.

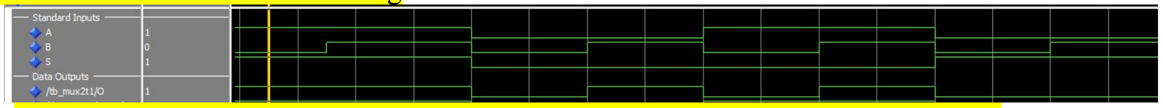
a)

a	b	s	f
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1

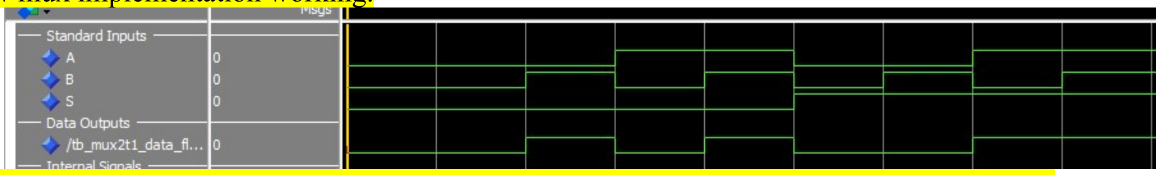
$f = \bar{s}b + sa$



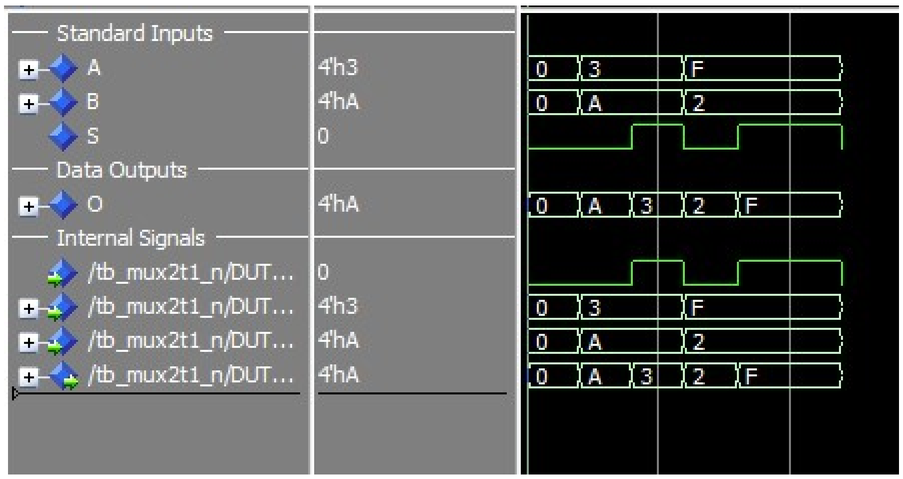
[Part 3.d] In your lab report, include a screenshot of the waveform. Make sure to label the screenshot with which module it is testing.



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

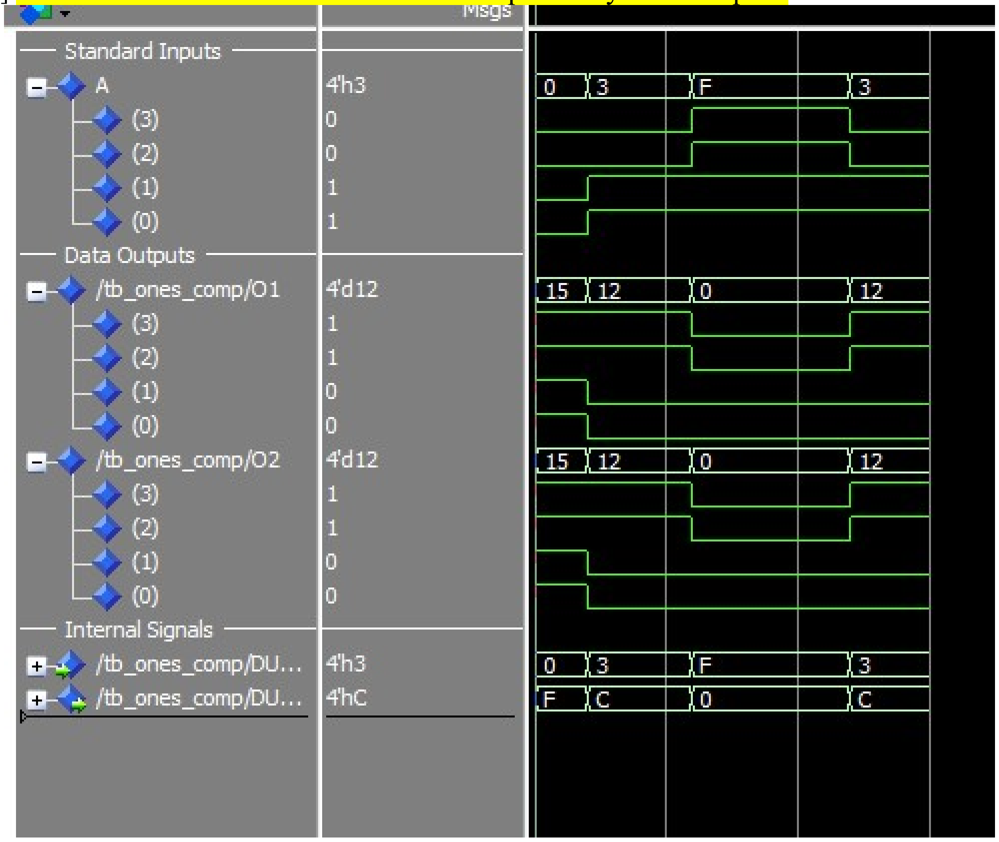


[Part 4] Include a waveform screenshot and corresponding description demonstrating it is working correctly.

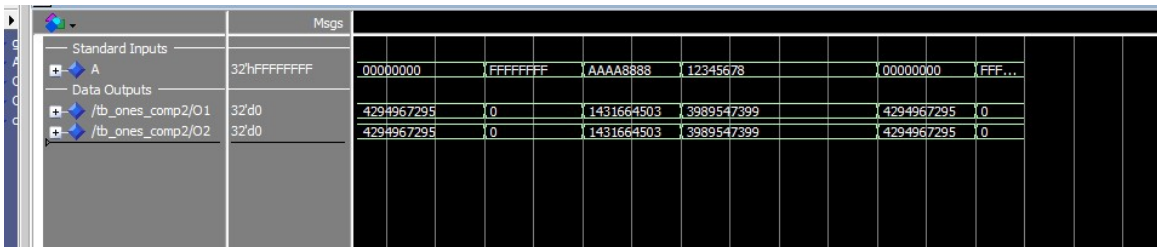


What is seen in this photo is the input to a test bench and the output. The output selects the A input if S is high, and the B input if the S is low, which is as described in the design.

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



In this first photo, I am showing a 4bit test where it is obvious that each bit is being inverted by both O1 and O2, where O1 is the structural ones, bit complement and O2 is the behavioral ones complement.



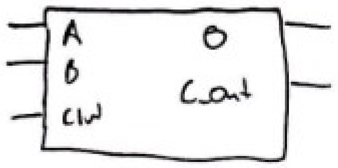
```
A                    32'hFFFFFFFF   00000000    FFFFFFFF  AAAA8888    12345678    00000000    FFF...
/tb_ones_comp2/O1    32'd0          4294967295  0         1431664503  3989547399  4294967295  0
/tb_ones_comp2/O2    32'd0          4294967295  0         1431664503  3989547399  4294967295  0
```

In this second photo, I decide to take things up a notch and to try a 32 bit number and as can be seen, they still have the same output.

[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.

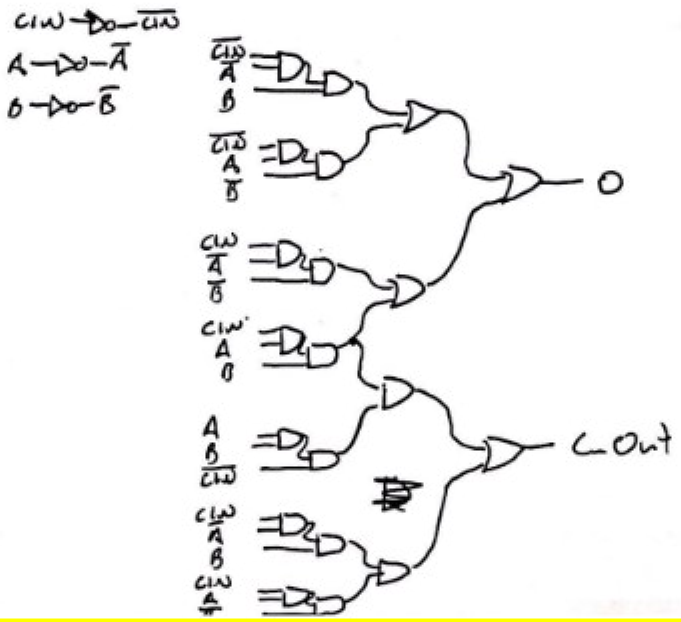
Full Adder

CIN	A	B	O	COut
0	0	0	0	0
0	0	1	1	0
0	1	0	1	
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

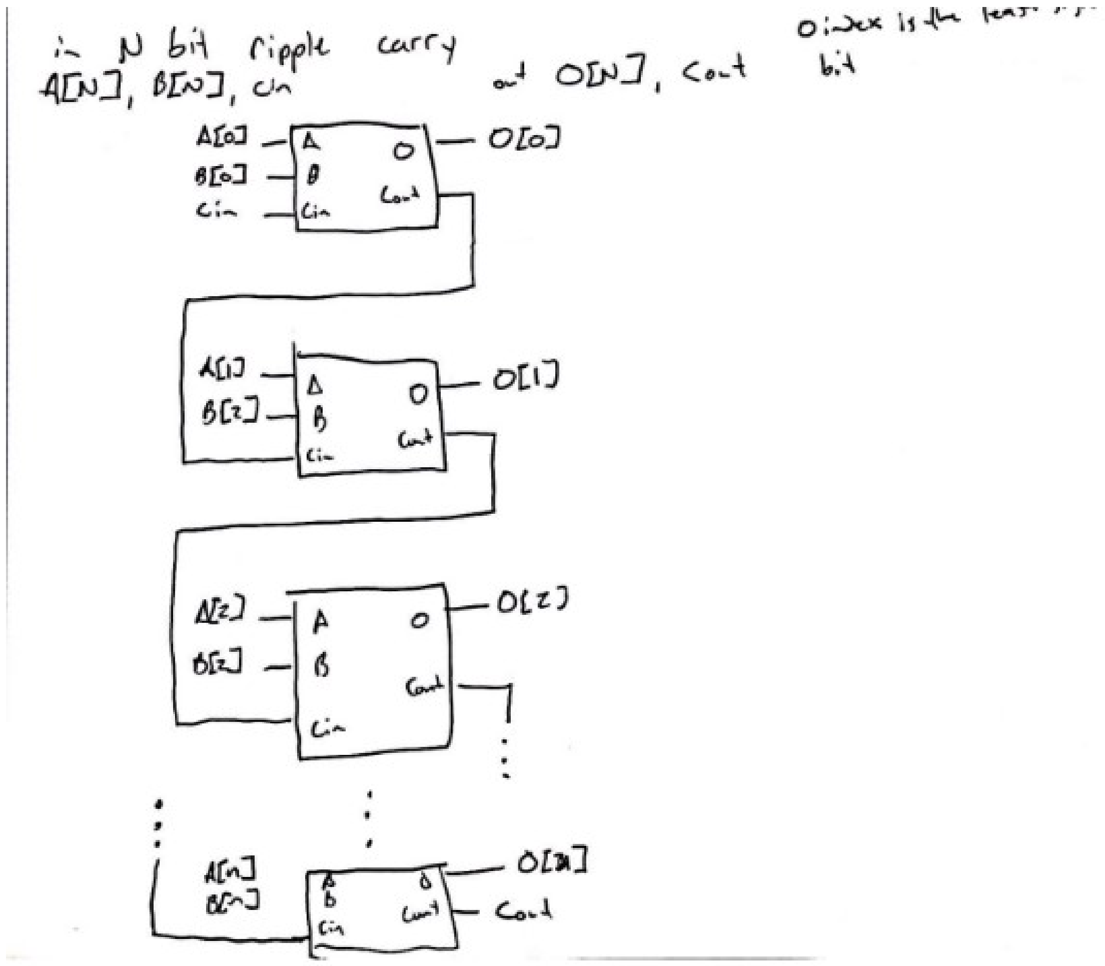


$O = \overline{CIN}\,\bar{A}B + \overline{CIN}\,A\bar{B} + CIN\,\bar{A}\bar{B} + CINA$

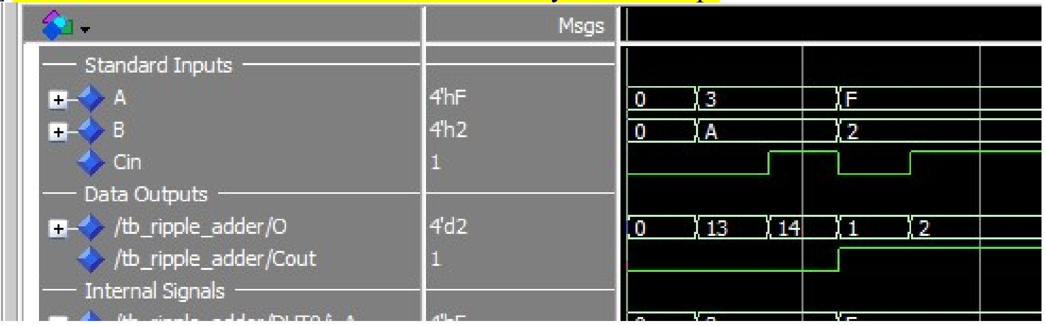
$COut = AB\overline{CIN} + CIN\,\bar{A}B + CINA\bar{B} + CINA$



[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.



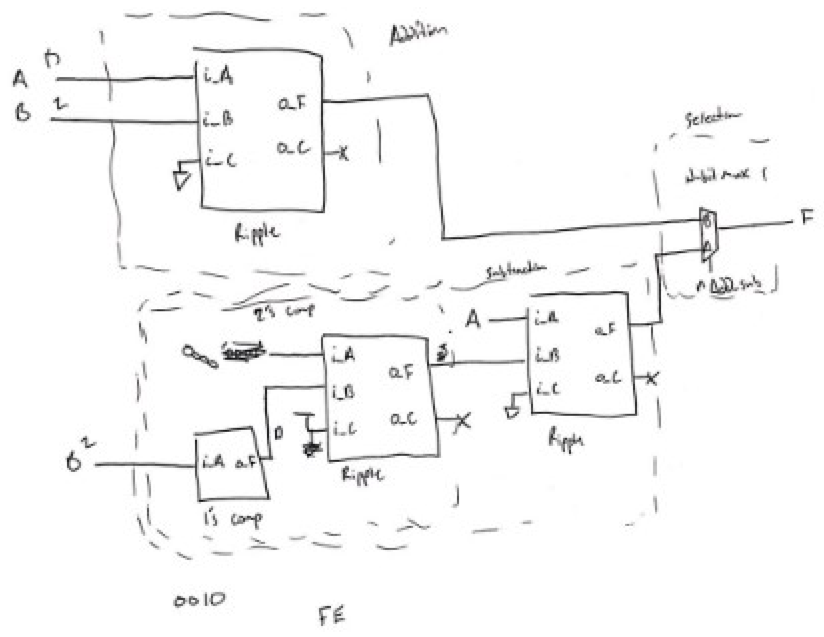
[Part 6.d] Include an annotated waveform screenshot in your write-up.



I'm not sure what is meant by annotated, other than describing what is going on. The output is the sum of the A and B inputs + the Cin input, and also has a Cout input.

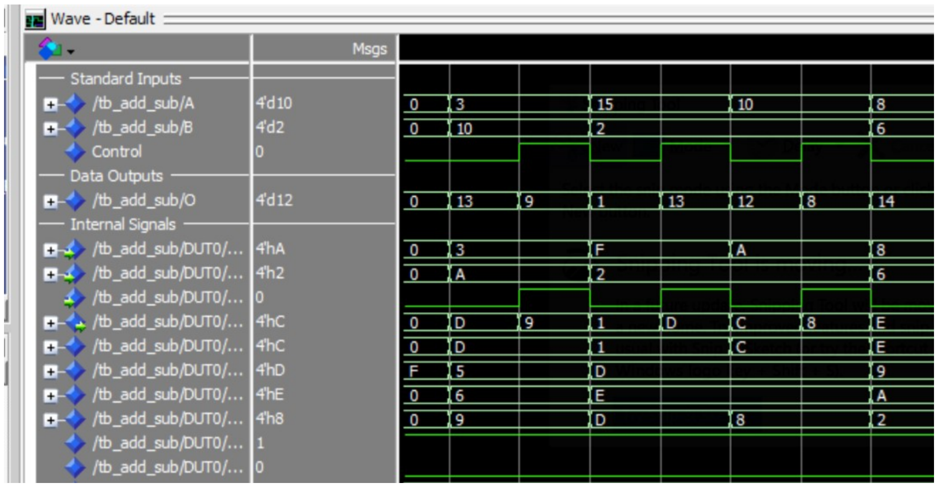
It shows the test case 3+A+0 -> 3+10+0=13. It shows and output of 13. It also shows the test case 3+A+1 which equals 14. Also it shows F+2+0=1 with a carry out bit. This represents 0x11 which is 17, which is correct. It also shows F+2+1=2 with a carry out bit, or 0x12 which is 18, which is correct.

[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.

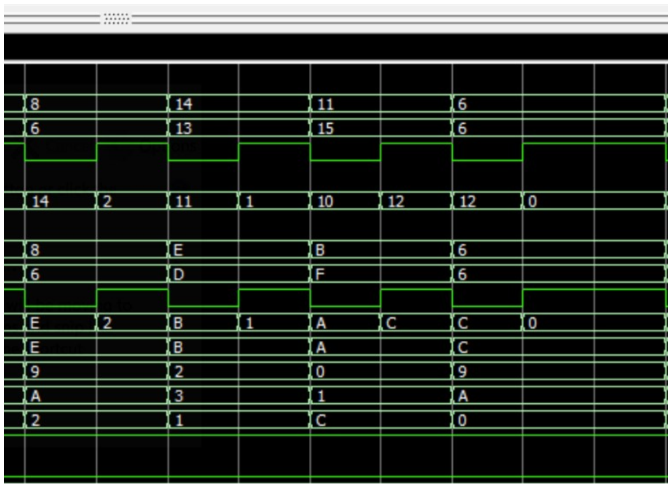


The nAdd_Sub is used to select if the user would like to use the addition mode or the subtraction mode. If it is set to 0, addition will be used, otherwise subtraction will be used.

[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?



```
Wave - Default
                                Msgs
— Standard Inputs —
/tb_add_sub/A          4'd10    0  3   15   10  8
/tb_add_sub/B          4'd2     0  10  2        6
Control                0
— Data Outputs —
/tb_add_sub/O          4'd12    0  13  9  1  13  12  8  14
— Internal Signals —
/tb_add_sub/DUT0/...   4'hA     0  3   F    A   8
/tb_add_sub/DUT0/...   4'h2     0  A   2        6
/tb_add_sub/DUT0/...   0
/tb_add_sub/DUT0/...   4'hC     0  D  9  1  D  C  8  E
/tb_add_sub/DUT0/...   4'hC     0  D   1    C   E
/tb_add_sub/DUT0/...   4'hD     F  5   D        9
/tb_add_sub/DUT0/...   4'hE     0  6   E        A
/tb_add_sub/DUT0/...   4'h8     0  9   D    8   2
/tb_add_sub/DUT0/...   1
/tb_add_sub/DUT0/...   0
```



I included multiple test cases of various numbers being added together and subtracted to try and find as many cases as I could to see that this worked in. I used 14 test cases where I added and subtracted 4 bit numbers.

```
----------------------------------------------
-- Test case 1:
-- 2 random 4 bit integers, S=0
A              <= "0011";
B              <= "1010";
add_sub_sig        <= '0';
wait for gCLK_HPER*2;

-- Test case 2:
-- 2 random 4 bit integers, S=1
A              <= "0011";
B              <= "1010";
add_sub_sig        <= '1';
wait for gCLK_HPER*2;
----------------------------------------------
-- Test case 3:
-- 2 more random 4 bit integers, S=0
A              <= "1111";
B              <= "0010";
add_sub_sig        <= '0';
wait for gCLK_HPER*2;

-- Test case 4:
-- 2 random 4 bit integers, S=1
A              <= "1111";
B              <= "0010";
add_sub_sig        <= '1';
wait for gCLK_HPER*2;
----------------------------------------------
```

```vhdl
    -- Test case 5:
    -- 2 more random 4 bit integers, S=0
    A              <= "1010";
    B              <= "0010";
    add_sub_sig        <= '0';
    wait for gCLK_HPER*2;

    -- Test case 6:
    -- 2 random 4 bit integers, S=1
    A              <= "1010";
    B              <= "0010";
    add_sub_sig        <= '1';
    wait for gCLK_HPER*2;
    ----------------------------------------------
    -- Test case 7:
    -- 2 more random 4 bit integers, S=0
    A              <= "1000";
    B              <= "0110";
    add_sub_sig        <= '0';
    wait for gCLK_HPER*2;

    -- Test case 8:
    -- 2 random 4 bit integers, S=1
    A              <= "1000";
    B              <= "0110";
    add_sub_sig        <= '1';
    wait for gCLK_HPER*2;
    ----------------------------------------------
    -- Test case 9:
    -- 2 more random 4 bit integers, S=0
    A              <= "1110";
    B              <= "1101";
    add_sub_sig        <= '0';
    wait for gCLK_HPER*2;

    -- Test case 10:
    -- 2 random 4 bit integers, S=1
    A              <= "1110";
    B              <= "1101";
    add_sub_sig        <= '1';
    wait for gCLK_HPER*2;
    ----------------------------------------------
    -- Test case 11:
    -- 2 more random 4 bit integers, S=0
    A              <= "1011";
    B             <= "1111";
    add_sub_sig       <= '0';
    wait for gCLK_HPER*2;

    -- Test case 12:
    -- 2 random 4 bit integers, S=1
    A             <= "1011";
    B             <= "1111";
    add_sub_sig       <= '1';
    wait for gCLK_HPER*2;
    ---------------------------------------------
    -- Test case 13:
    -- 2 more random 4 bit integers, S=0
    A             <= "0110";
    B             <= "0110";
    add_sub_sig       <= '0';
    wait for gCLK_HPER*2;

    -- Test case 14:
    -- 2 random 4 bit integers, S=1
    A             <= "0110";
    B             <= "0110";
    add_sub_sig       <= '1';
    wait for gCLK_HPER*2;
```