EE496 Computer Architecture and VHDL

Assignment 1: Adder/Subtractor

Ciaran O'Donnell

Student Number: 15414048

1. One-bit full adder (FA)

1.1 Full adder VHDL model

The first step of this lab was to write a VHDL model for a full adder. I constructed my adder with the following code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          ci : in STD_LOGIC;
          co : out STD_LOGIC;
          s : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is
  signal tmp: std_logic;

begin
      tmp <= a xor b after 1 ns;
      co <= (a and b) or (ci and tmp) after 2 ns;
      s <= ci xor tmp after 1 ns;
end Behavioral;
```

This code allowed to have two one-bit inputs (a and b) as well as a carry in (ci) as declared above as well as two outputs sum (s) and carry out (co). I would also need an intermediate signal in my model, I declared this as tmp.
The logic for a full adder is:

Sum = (A XOR B) XOR Cin
Cout = A*B + Cin*(A XOR B)

Tmp was used as the A XOR B variable, so the logic in vhdl model became:

```
tmp <= a xor b after 1 ns;
co <= (a and b) or (ci and tmp) after 2 ns;
s <= ci xor tmp after 1 ns;
```

This model was then ready to compile.

1.2 Test bench for full adder

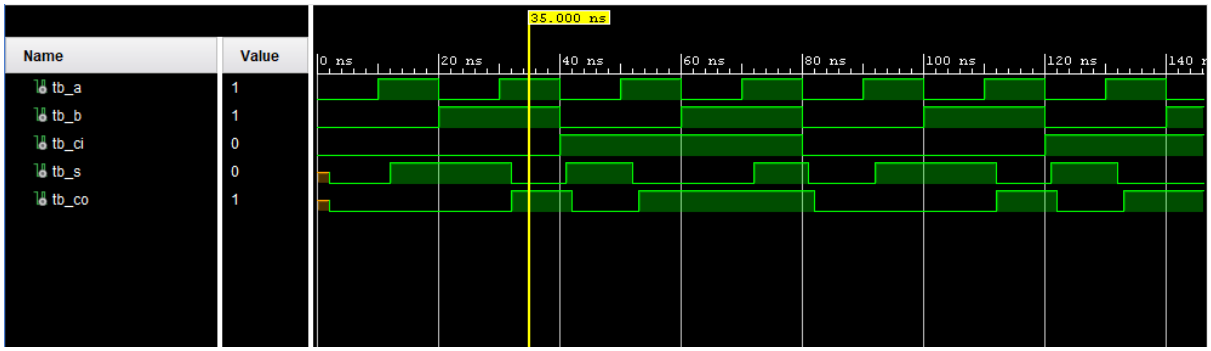
The next step was to write a VHDL test bench in order to run simulations on the full-adder. As the adder adds a two one bit numbers and accepts a carry in this means there is a total of 8 possible inputs for the model. I made a simple test bench that would step up through the inputs after pre-determined timesteps:

```
entity tb_full_adder is
end tb_full_adder;
architecture Behavioral of tb_full_adder is
 signal tb_a, tb_b, tb_ci, tb_s, tb_co: std_logic := '0';

 component full_adder is
      port (a: in std_logic;
           b: in std_logic;
           ci: in std_logic;
           s:  out std_logic;
           co: out std_logic);
 end component;
begin
 uut: full_adder
     port map (a => tb_a,
              b => tb_b,
              ci => tb_ci,
              s => tb_s,
              co => tb_co);

  tb_a <= not tb_a after 10 ns;
  tb_b <= not tb_b after 20 ns;
  tb_ci <= not tb_ci after 40 ns;
end Behavioral;
```

This code then stepped up through each input, changing tb_a which was linked to a every 10ns, tb_b which was linked to b every 20ns and finally tb_ci which was linked to ci every 30ns. This gave all possible inputs to the system over a span of 80ns.

1.3 Simulation of Full Adder

Here is to simulation output for the model above, it can be seen that all of combinations of inputs are present in the first 80ns and that all inputs give a correct output.



The section highlighted shows a=1, b-1, ci=0, and therefore sum=0, co=1.

2. 4-bit adder/subtractor

2.1 4-bit adder/subtractor using full adder component

Again, the first task was to write a vhdl model for the adder subtractor. This was done by using the full adder as a component and using four of these, one for each order of bit. The carry out from each was used as the carry in of the next. This gave a 4-bit adder. However, to implement a subtractor twos complement was used, this was done by adding an input that when 1 would be input to the least significant bit adder (+1) and XOR with all inputs of the second input number (flipping its bits). This would give the required answer.

```
entity adder_subtractor is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          sub : in STD_LOGIC;
          sum : out STD_LOGIC_VECTOR (3 downto 0);
          Co : out STD_LOGIC);

end adder_subtractor;

architecture Behavioral of adder_subtractor is

component full_adder
port(   a, b, ci : in STD_LOGIC;
        s, co: out STD_LOGIC);
end component;
signal c: std_logic_vector(3 downto 0);
signal Bin: std_logic_vector(3 downto 0);
begin

Bin(0) <= b(0) xor sub;
Bin(1) <= b(1) xor sub;
Bin(2) <= b(2) xor sub;
Bin(3) <= b(3) xor sub;

   FA0: full_adder port map (A(0), Bin(0), sub, sum(0), c(1));
   FA1: full_adder port map (A(1), Bin(1), c(1), sum(1), c(2));
   FA2: full_adder port map (A(2), Bin(2), c(2), sum(2), c(3));
   FA3: full_adder port map (A(3), Bin(3), c(3), sum(3), c(0));

Co <= c(0);

end Behavioral;
```

I did this by using a signal Bin that equalled B XOR Sub as the input for B in each Adder. As well as having the first adders carry in equal sub. And the Carry out of the entire model was equal to the carry out of the most significant bits adder.

2.2 Test bench for adder subtractor

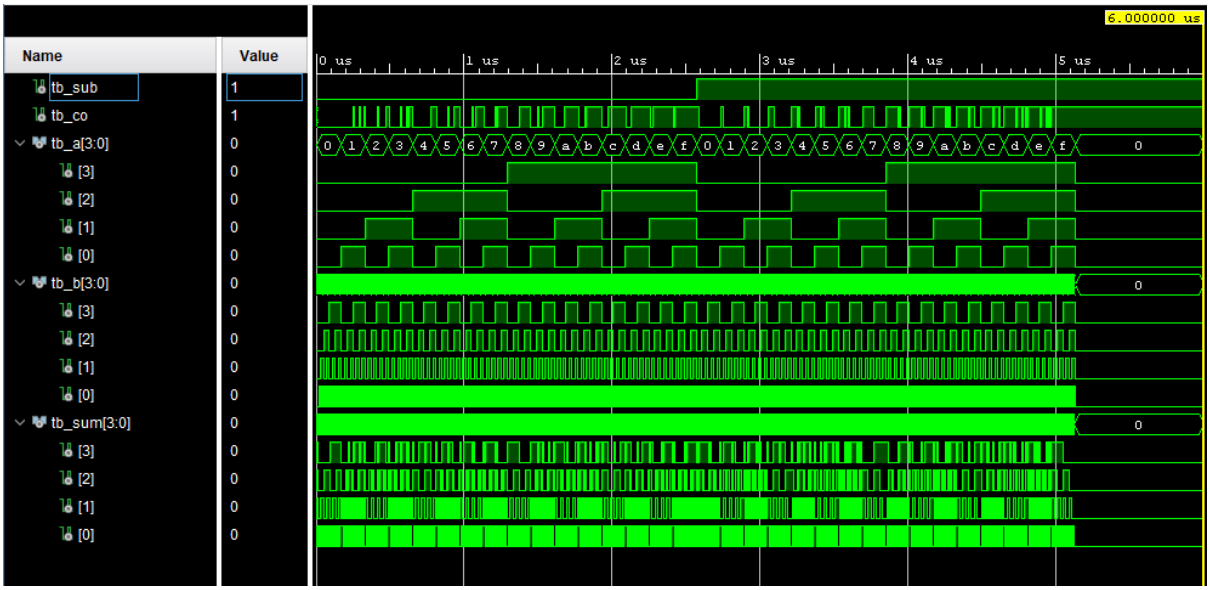
The next step was to implement a test bench for this model. I did this using for loops. To test all potential inputs there would be 16*16*2 possible inputs. I structured my testbench to step up through every value A, and for each value of A step through every value of B and wait a period, then when all values of a are complete turn on the subtractor and do the same again now subtracting for every value. Using twos complement however means we can only have a range of -8 to 7 as the most significant bit is used as the sign bit.

```
entity adder_subtractor_tb is
end adder_subtractor_tb;

architecture Behavioral of adder_subtractor_tb is

signal tb_sub, tb_co: std_logic := '0';
signal tb_a, tb_b, tb_sum: std_logic_vector (3 downto 0);

 component adder_subtractor is
       port  (A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            sub : in STD_LOGIC;
            sum : out STD_LOGIC_VECTOR (3 downto 0);
            Co : out STD_LOGIC);
 end component;
begin
 uut: adder_subtractor
      port map (A => tb_a,
             B => tb_b,
             sub => tb_sub,
             sum => tb_sum,
             Co => tb_co);
process
begin
tb_a <= "0000";
tb_b <= "0000";
wait for 10ns;

sub_loop: for K in 0 to 1 loop
      a_loop : for I in 0 to 15 loop
        b_loop : for J in 0 to 15 loop
              wait for 10ns;
           tb_b <= std_logic_vector( unsigned(tb_b) + 1 );
        end loop b_loop;
        tb_a <= std_logic_vector( unsigned(tb_a) + 1 );
      end loop a_loop;
      tb_sub <= '1';
end loop sub_loop;
wait for 1000ns;
end process;
end Behavioral;
```

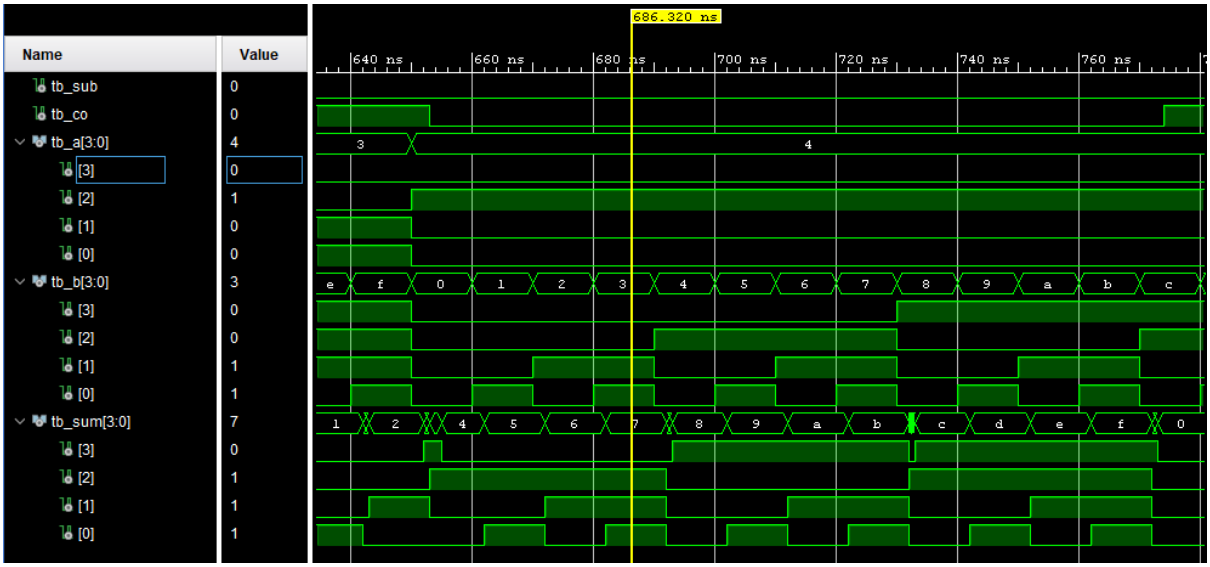
This gave an output for every possible value input for a time of 10ns.

2.3 Simulation result for adder subtractor

So then when the simulation was ran the following output was received:



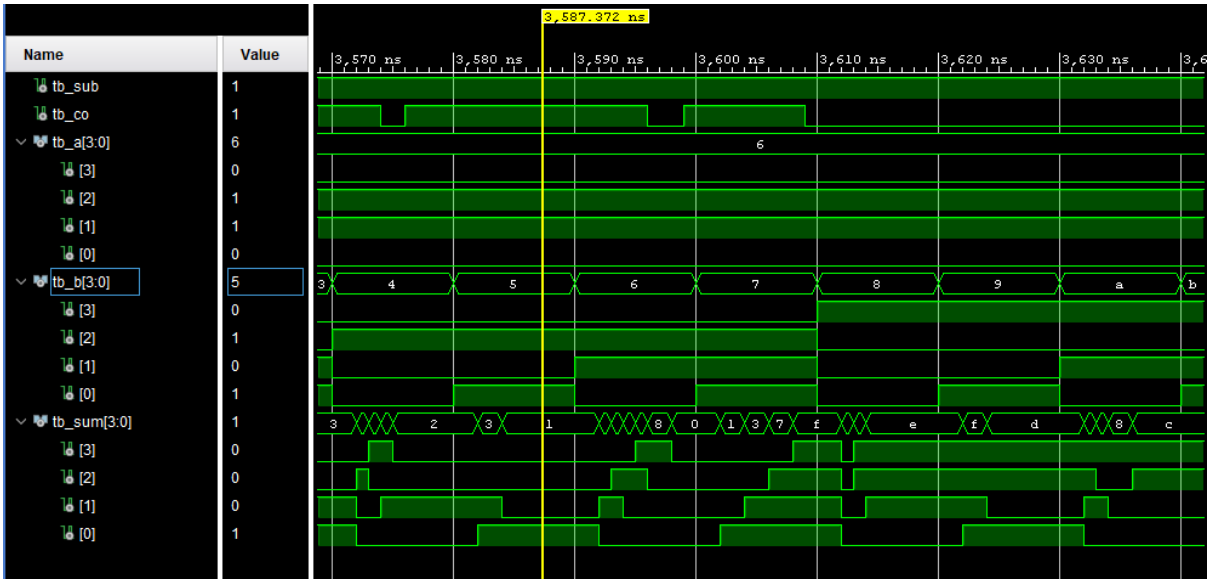
We are currently too zoomed out to see great detail in the results however from this view we can see that A steps up in value from 0 to 15 and then the sub variable goes to 1 and A steps up from 0 to 15 again.



Once we zoom in we can see that the adder is working correctly and that B is also stepping up from 0 to 15 for each value of A. Highlighted we can see the output of:

	4	0100
+	3	0011
=	7	0111

When we move up to see subtracted numbers in our simulation this is what we can see:



The highlighted portion shows:

	6	0110
-	5	0101
=	1	0001

However, looking at the output for sum we can see a long propagation delay as each adder along from least significant bit is delayed by the delay of the adder before it due to using its carry as well as the delay of the XOR gate on the input B.

We can also see in this image 6-8=E

	6	0110			6	0110
-	8	1000	is same as	+	-8	1000
					E	1101
			Which is the twos complement value			-2

When subtracting and the answer is a negative the answer giving will have a leading one to show that it is a negative number and that its value is a twos complement number.

3 Flags for Adder Subtractor

3.1 Negative Flag

To implement a negative flag the most significant output bit was used, if this bit was '1' the Negative flag was positive,

i.e. N <= sum(3);

for this to work (and also for the zeros flag) I made the sum bus into an inout vector.

3.2 A Zero Flag

To implement a zero flag the sum out put were all OR and then the answer was inverted, so if any output value was 1 the zero flag would be 0

Z <= not(sum(0) OR sum(1) OR sum(2) OR sum(3));

3.3 Carry Flag

This was simply the signal from the output of the final adders Co

C <= cry(0);

cry(0) is the Carry out of my most significant bit adder in my personal vhdl model.

3.4 Overflow Flag

The overflow flag was implemented by checking if the most significant bit was high, and it is not a subtraction it has overflown, or if it is a subtraction and the most significant bit is not 1.

V <= (sum(3) and (not sub)) OR (sub and (not sum(3)));

Here is the entire model with these bits of code included:

```
entity adder_subtractor is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          sub : in STD_LOGIC;
          sum : inout STD_LOGIC_VECTOR (3 downto 0);
          C : out STD_LOGIC;
          N : out STD_LOGIC;
          Z : out STD_LOGIC;
          V : out STD_LOGIC);

end adder_subtractor;

architecture Behavioral of adder_subtractor is

component full_adder
port(   a, b, ci : in STD_LOGIC;
        s, co: out STD_LOGIC);
end component;
signal cry: std_logic_vector(3 downto 0);
signal Bin: std_logic_vector(3 downto 0);
begin

Bin(0) <= b(0) xor sub;
Bin(1) <= b(1) xor sub;
```

```
Bin(2) <= b(2) xor sub;
Bin(3) <= b(3) xor sub;

   FA0: full_adder port map (A(0), Bin(0), sub, sum(0), cry(1));
   FA1: full_adder port map (A(1), Bin(1), cry(1), sum(1), cry(2));
   FA2: full_adder port map (A(2), Bin(2), cry(2), sum(2), cry(3));
   FA3: full_adder port map (A(3), Bin(3), cry(3), sum(3), cry(0));

C <= cry(0);
N <= sum(3);
Z <= not(sum(0) OR sum(1) OR sum(2) OR sum(3));
V <= (sum(3) and (not sub)) OR (sub and (not sum(3)));

end Behavioral;
```

3.5 Test bench for flags

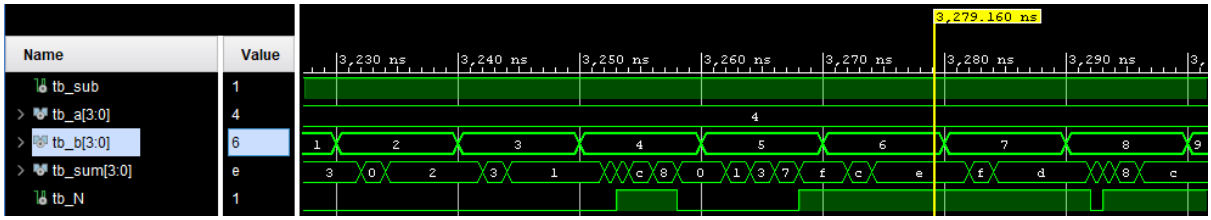
For this next task I used the previous test bench and simply included these variables to be output as I had already previously set up to test for every possible input:

```
entity adder_subtractor_tb is

end adder_subtractor_tb;

architecture Behavioral of adder_subtractor_tb is

signal tb_sub, tb_C, tb_Z, tb_V, tb_N: std_logic := '0';
signal tb_a, tb_b, tb_sum: std_logic_vector (3 downto 0);

 component adder_subtractor is
      port  (A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           sub : in STD_LOGIC;
           sum : inout STD_LOGIC_VECTOR (3 downto 0);
           C : out STD_LOGIC;
           N : out STD_LOGIC;
           Z : out STD_LOGIC;
           V : out STD_LOGIC);
 end component;


begin
 uut: adder_subtractor
     port map (A => tb_a,
           B => tb_b,
           sub => tb_sub,
           sum => tb_sum,
           C => tb_C,
           N => tb_N,
           V => tb_V,
           Z => tb_Z);
```

```
process
begin

tb_a <= "0000";
tb_b <= "0000";


wait for 10ns;

sub_loop: for K in 0 to 1 loop
        a_loop : for I in 0 to 15 loop
          b_loop : for J in 0 to 15 loop

              wait for 10ns;

            tb_b <= std_logic_vector( unsigned(tb_b) + 1 );
          end loop b_loop;

          tb_a <= std_logic_vector( unsigned(tb_a) + 1 );
        end loop a_loop;

        tb_sub <= '1';

end loop sub_loop;

wait for 1000ns;

end process;

end Behavioral;
```

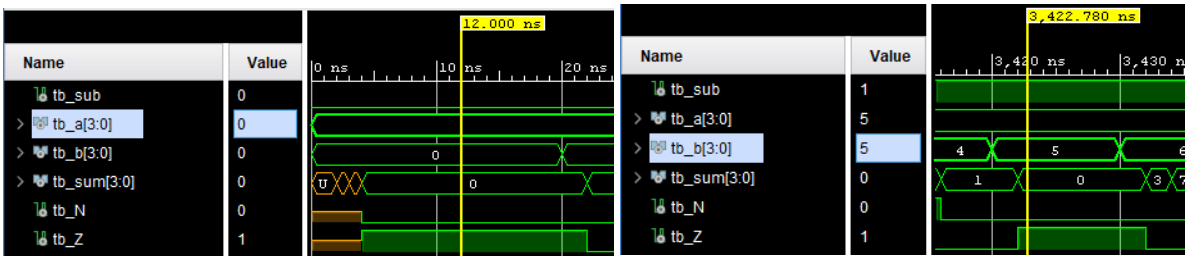
3.6 Simulation Results

Once the simulation was ran we can see that while in range all values in the negative set the negative flag to 1:



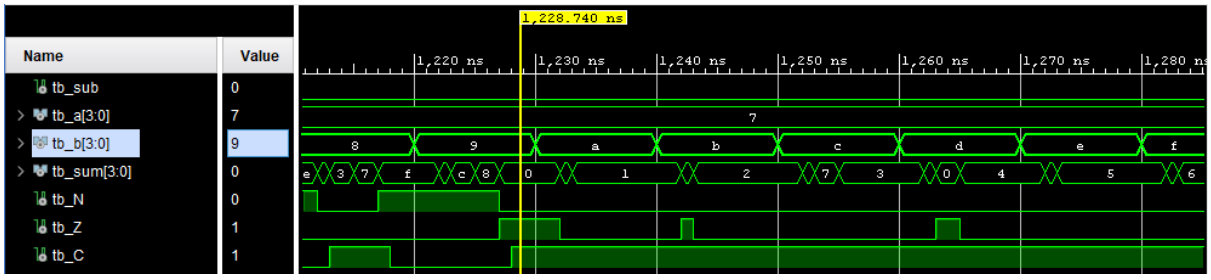
Here we can see with A equal to 4 at
4-1, 4-2, 4-3, 4-4 the N flag is not set once propagation delay has settled, but from 4-5 on the negative flag activates as expected.

Next to test was the zero flag:



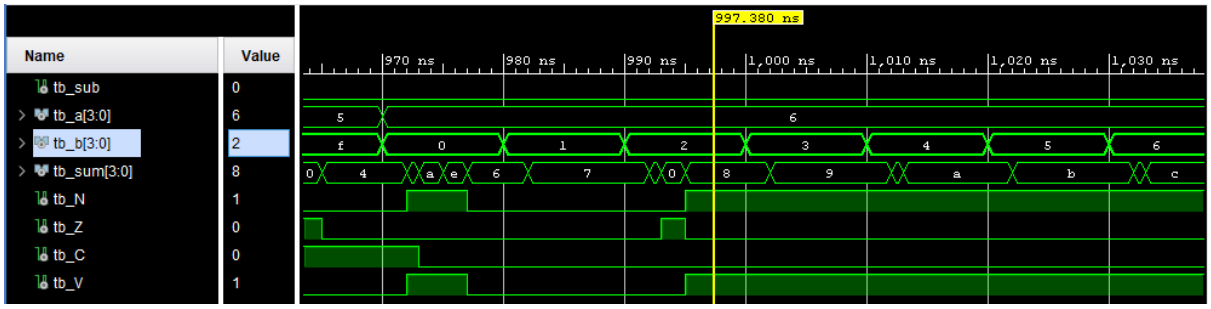
We can see here that at both 0+0 and 5-5 when the sum is equal to 0 the 0 flag is ‘1’ as expected.

Next test was of the Carry Flag:

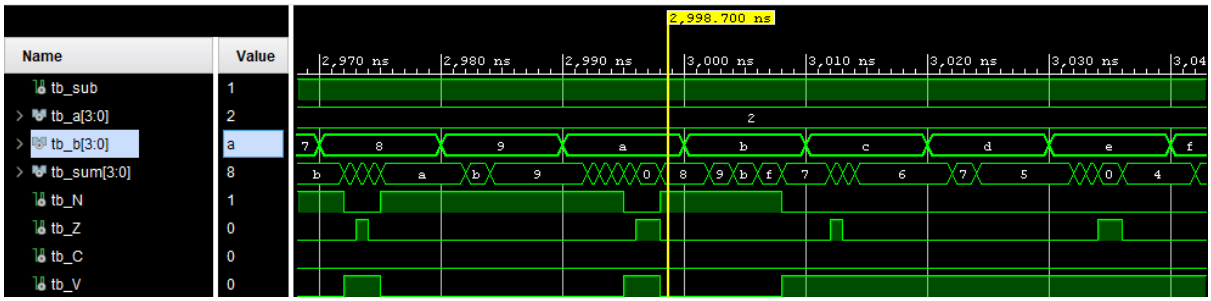


Here we can see that once we see that 7+8 correctly does not activate the carry flag but from 7+9 onwards the carry flag is activated.

Finally we must test the overflow flag:



Here we can see while the sum value is below 8 (6+0 and 6+1) the V flag is 0 but as soon as the sum value hits 8 the overflow flag is activated.



And here we can see when subtracting, up to 2-10 the overflow flag is not activated as -8 is within range but from 2-11 on the flag is activated.

Final Questions:

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
   This lab took me between 5 and 8 hours in total to complete.

2. Write a few sentences describing the purpose of this lab.
   The purpose of this lab was to learn the basics of writing and testing VHDL models, to learn some of the HDL design and testing process. As well as to revise some more basic logic theory.

3. Did your full adder pass tests for all eight possible inputs?
   Yes, my full adder passed for all possible inputs.

4. How did you test your 4-bit adder/subtractor?
   To test my 4 bit adder and subtractor I stepped through all possible inputs for B, then added one to A and stepped through all possible inputs for B again, once I had done this for all possible inputs for A I set the subtraction bit and di them all again.