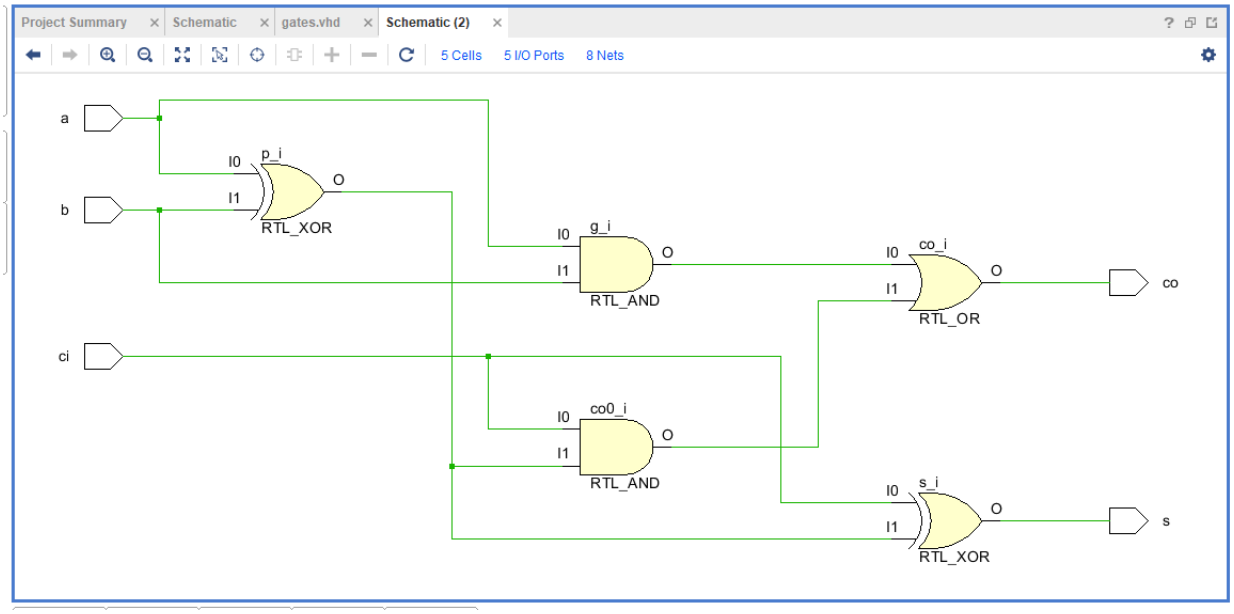
EE496 Computer Architecture and VHDL

Assignment 1: Adder/Subtractor
Due date: 25 October 2017

1.1Write a VHDL model for a one-bit full add

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

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

architecture Behavioral of gates is
  signal g: std_logic;
  signal p: std_logic;
begin
  p <= a xor b;
  g <= a and b;
  s <= ci xor p;
  co <= g or (ci and p);

end Behavioral;
```

RTL sketch generated by Vivado for the 1-bit full adder is illustrated below.



1.2Write a test bench for the one-bit full adder

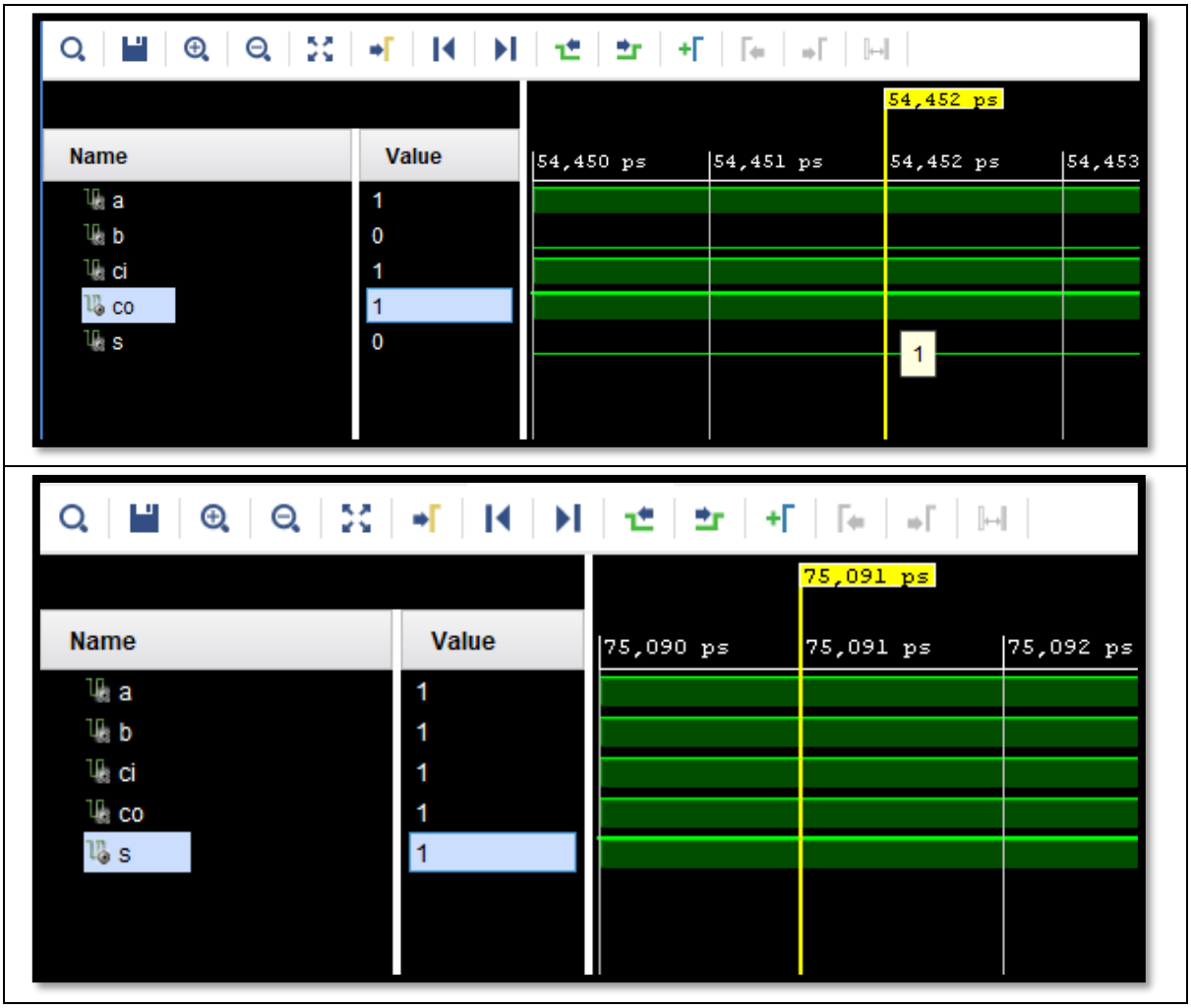
```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity gates_tb is
end;

architecture bench of gates_tb is

  component gates
     Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          ci : in STD_LOGIC;
          co : out STD_LOGIC;
          s : out STD_LOGIC);
  end component;

  signal a: STD_LOGIC;
  signal b: STD_LOGIC;
  signal ci: STD_LOGIC;
  signal co: STD_LOGIC;
  signal s: STD_LOGIC;

begin

  uut: gates port map ( a  => a,
                  b  => b,
                  ci => ci,
                  co => co,
                  s  => s );

  stimulus: process
  begin

    -- Put initialisation code here
      a  <= '0';
      b  <= '0';
      ci <= '0';
      wait for 10 ns;
       a  <= '0';
       b  <= '0';
       ci <= '1';
       wait for 10 ns;
       a  <= '0';
       b  <= '1';
```

```
        ci <= '0';
        wait for 10 ns;
        a  <= '0';
        b  <= '1';
        ci <= '1';
        wait for 10 ns;
        a  <= '1';
        b  <= '0';
        ci <= '0';
        wait for 10 ns;
        a  <= '1';
        b  <= '0';
        ci <= '1';
        wait for 10 ns;
        a  <= '1';
        b  <= '1';
        ci <= '0';
        wait for 10 ns;
        a  <= '1';
        b  <= '1';
        ci <= '1';
        wait for 10 ns;
        a  <= '0';
        b  <= '0';
        ci <= '0';
        wait for 10 ns;
    -- Put test bench stimulus code here

    wait;
  end process;


end;
```

1.3Present the simulation results of the one-bit full adder using the test bench

INPUT A	INPUT B	CARRYIN Ci	SUM S	CARRYOUT CO
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

Results from simulation correspond to the 1-bit full adder truth table and can be confirmed below.



2.1 Use the 4-bit adder as component to build a 4-bit adder/substractor (hint: A minus B equals A plus the two's complement of B, i.e. A – B = A + (invert of B) + 1).

To complete this section, the code was split into the following 2 parts.

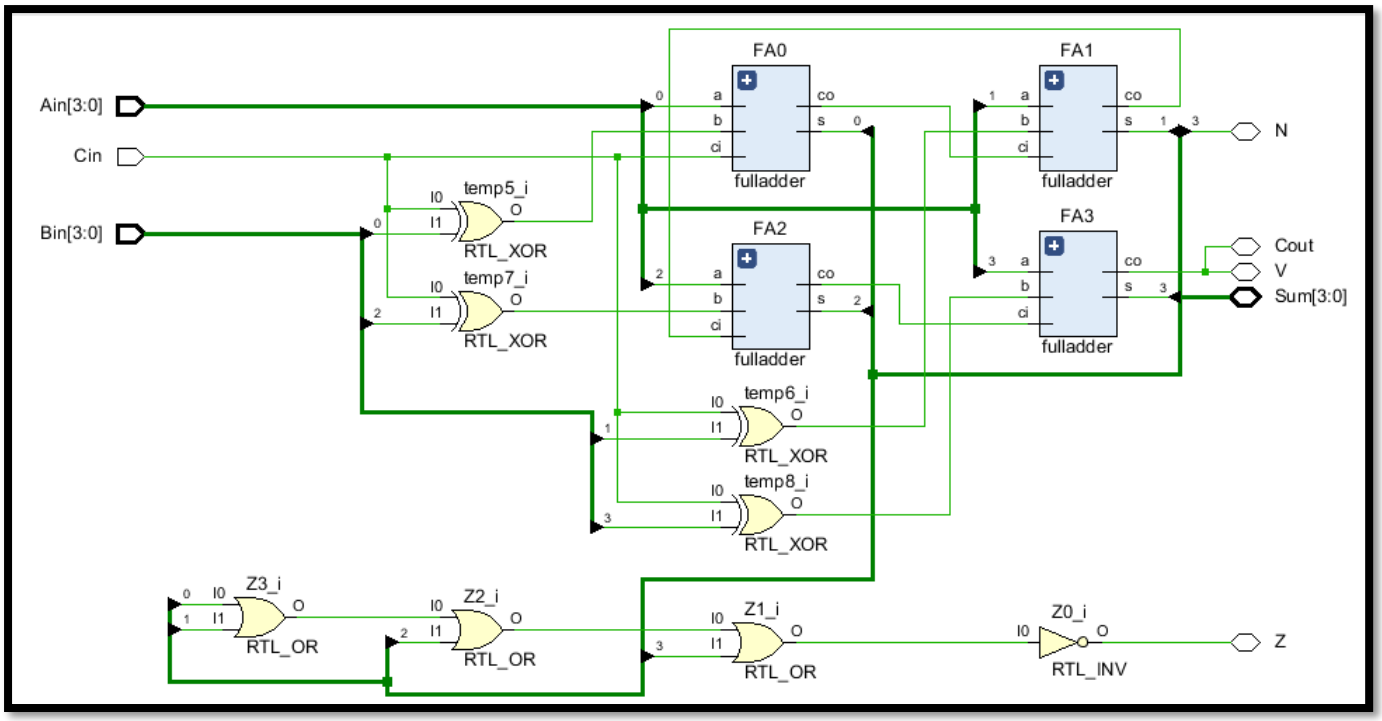
2.1.1 1-bit full adder (which was the same as part 1 of the assignment)
2.1.2 The final part is a code template that combines the one-bit adder and the xor of input Bin and creates 4 of them in series.

Code can be found here:

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

entity adder_subtractor is
    Port ( Ain : in STD_LOGIC_VECTOR (3 downto 0);
```

```
        Bin : in STD_LOGIC_VECTOR (3 downto 0);
        Cin : in STD_LOGIC;
        Cout : inout STD_LOGIC;
        Sum : inout STD_LOGIC_VECTOR (3 downto 0);
        Z,N,V,C : inout STD_LOGIC);
end adder_subtractor;

architecture Behavioral of adder_subtractor is
   signal  temp1, temp2, temp3, temp4, temp5, temp6, temp7, temp8, temp9, temp10,
temp11:std_logic;

begin

   temp5 <= Cin xor Bin(0);
   temp6 <= Cin xor Bin(1);
   temp7 <= Cin xor Bin(2);
   temp8 <= Cin xor Bin(3);

   FA0 : entity work.fulladder port map(a => Ain(0),b => temp5,ci => Cin,s => Sum(0),co =>
temp9);
   FA1 : entity work.fulladder port map(a => Ain(1),b => temp6,ci => temp9,s => Sum(1),co
=> temp10);
   FA2 : entity work.fulladder port map(a => Ain(2),b => temp7,ci => temp10,s =>
Sum(2),co => temp11);
   FA3 : entity work.fulladder port map(a => Ain(3),b => temp8,ci => temp11,s =>
Sum(3),co => Cout);

   --negative flag code
   N <= Sum(3);
   -- zero flag code
   Z <= not (Sum(0) or Sum(1) or Sum(2) or Sum(3));
   -- carry flag
   V <= Cout;
  --overflow
   V <=  temp11 xor Cout;
end Behavioral;
```

The schematic sketch can be seen below:



2.2 Write a test bench for the 4-bit adder/substractor

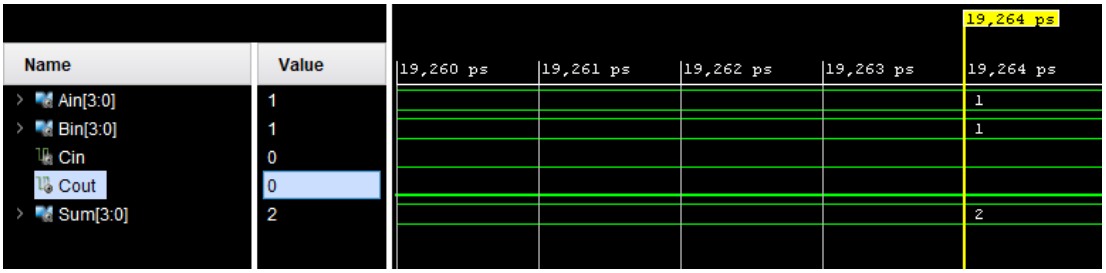
```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity adder_subtractor_tb is
end;

architecture bench of adder_subtractor_tb is

 component adder_subtractor
    Port ( Ain : in STD_LOGIC_VECTOR (3 downto 0);
           Bin : in STD_LOGIC_VECTOR (3 downto 0);
           Cin : in STD_LOGIC;
           Cout : inout STD_LOGIC;
           Sum : inout STD_LOGIC_VECTOR (3 downto 0));
          -- Z,N,V : inout STD_LOGIC);
 end component;

 signal Ain: STD_LOGIC_VECTOR (3 downto 0);
 signal Bin: STD_LOGIC_VECTOR (3 downto 0);
 signal Cin: STD_LOGIC;
 signal Cout: STD_LOGIC;
 signal Sum: STD_LOGIC_VECTOR (3 downto 0);
 --signal Z,V,N : STD_LOGIC;
begin

 uut: adder_subtractor port map ( Ain  => Ain,
                                  Bin  => Bin,
```

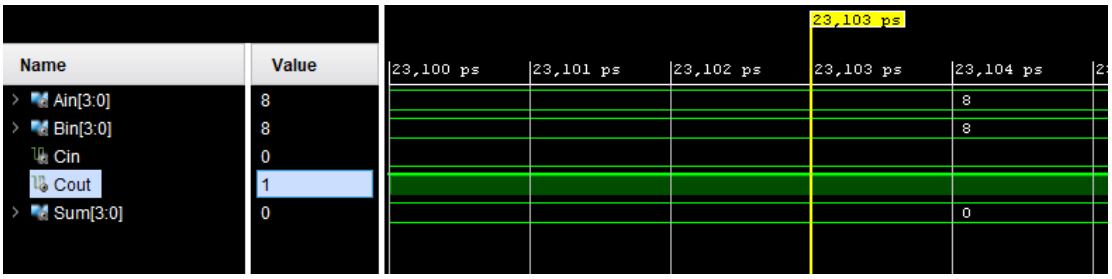
```
                        Cin  => Cin,
                        Cout => Cout,
                        Sum  => Sum);
                        --Z => Z,
                       -- V => V,
                       -- N => N);

stimulus: process

begin

  -- Put initialisation code here
  -- addition
     Ain <= "0000";
     Bin <= "0000";
     Cin <= '0';
     wait for 10 ns;
     Ain <= "0001";
     Bin <= "0001";
     Cin <= '0';
     wait for 10 ns;
     Ain <= "1000";
     Bin <= "1000";
     Cin <= '0';
     wait for 10 ns;
     Ain <= "0100";
     Bin <= "0011";
     Cin <= '0';
     wait for 10 ns;
     Ain <= "1111";
     Bin <= "0000";
     Cin <= '0';
     wait for 10 ns;
  -- subtraction
     Ain <= "0000";
     Bin <= "0000";
     Cin <= '1';
     wait for 10 ns;
     Ain <= "0001";
     Bin <= "0001";
     Cin <= '1';
     wait for 10 ns;
     Ain <= "1000";
     Bin <= "1000";
     Cin <= '1';
     wait for 10 ns;
     Ain <= "0100";
     Bin <= "0011";
     Cin <= '1';
     wait for 10 ns;
     Ain <= "1111";
```

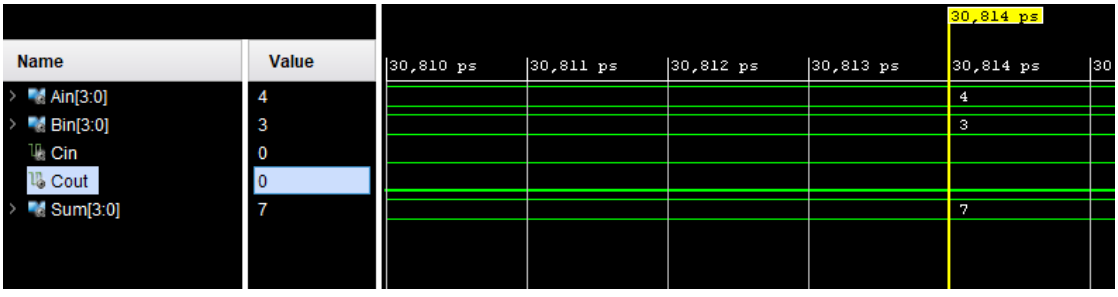
```
        Bin <= "0000";
        Cin <= '1';
        wait for 10 ns;


    -- Put test bench stimulus code here

    wait;
  end process;


end;
```

2.3 Present the simulation results of the 4-bit adder/substractor using the test bench

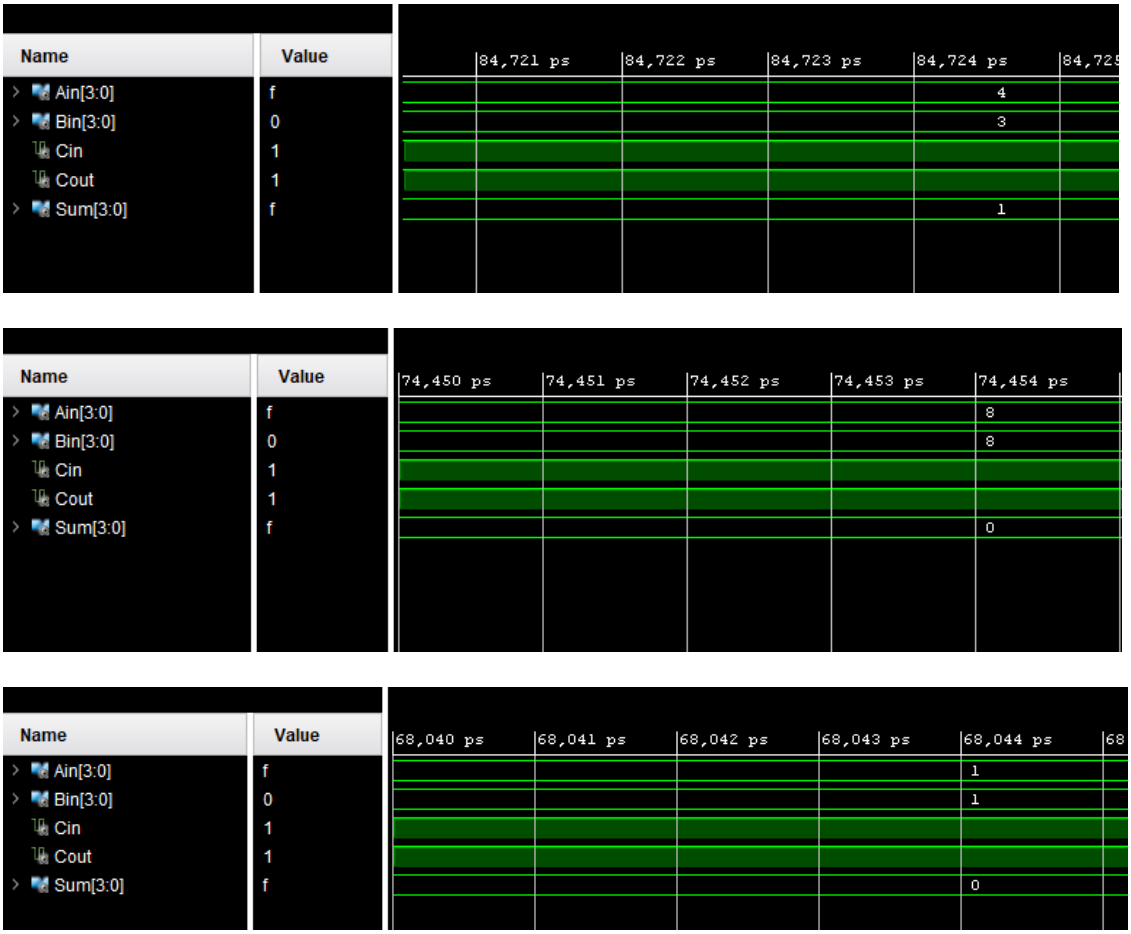
ADDITION







SUBTRACTION



3.1 Negative flag (N), which indicates the result is negative (i.e. most significant bit is 1)

3.2 A zero flag (Z), which indicates that the result is 0 (i.e. all bits of the result are 0).

3.3 Carry Flag (C), which indicates the adder/subtractor produced a carry, i.e. overflow in unsigned arithmetic. It is also used for propagating the carry between words in multipleprecision arithmetic.

3.4 4 Overflow flag (V), which indicates that the result of the addition or subtraction of two two's complement numbers overflowed (i.e. outside the range of numbers that can be represented using a 4-bit two's complement number, i.e. outside the range of -8 to 7 in this case).

```
--negative flag code
N <= Sum(3);
-- zero flag code
Z <= not (Sum(0) or Sum(1) or Sum(2) or Sum(3));
-- carry flag
V <= Cout;
--overflow
V <=  temp11 xor Cout;
```

Above is the code for part 3.1 to 3.4

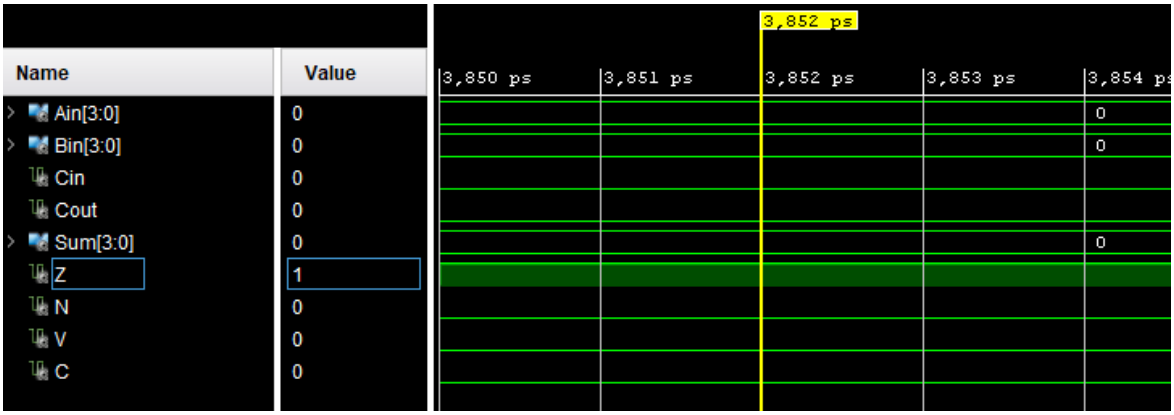
3.5 Write a test bench for the 4-bit adder/substractor extended with output flags

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity adder_subtractor_tb is
end;

architecture bench of adder_subtractor_tb is

  component adder_subtractor
     Port ( Ain : in STD_LOGIC_VECTOR (3 downto 0);
          Bin : in STD_LOGIC_VECTOR (3 downto 0);
          Cin : in STD_LOGIC;
          Cout : inout STD_LOGIC;
          Sum : inout STD_LOGIC_VECTOR (3 downto 0));
          Z,N,V,C : inout STD_LOGIC);
  end component;

  signal Ain: STD_LOGIC_VECTOR (3 downto 0);
  signal Bin: STD_LOGIC_VECTOR (3 downto 0);
  signal Cin: STD_LOGIC;
  signal Cout: STD_LOGIC;
  signal Sum: STD_LOGIC_VECTOR (3 downto 0);
  signal Z,N,V,C : STD_LOGIC;
begin

  uut: adder_subtractor port map ( Ain  => Ain,
                           Bin  => Bin,
                           Cin  => Cin,
                           Cout => Cout,
                           Sum  => Sum),
                           Z => Z,
                           N => N,
                           V => V,
                           C => C);

  stimulus: process

  begin

    -- Put initialisation code here
    -- addition
       Ain <= "0000";
       Bin <= "0000";
       Cin <= '0';
       wait for 10 ns;
       Ain <= "0001";
       Bin <= "0001";
       Cin <= '0';
       wait for 10 ns;
       Ain <= "1000";
```

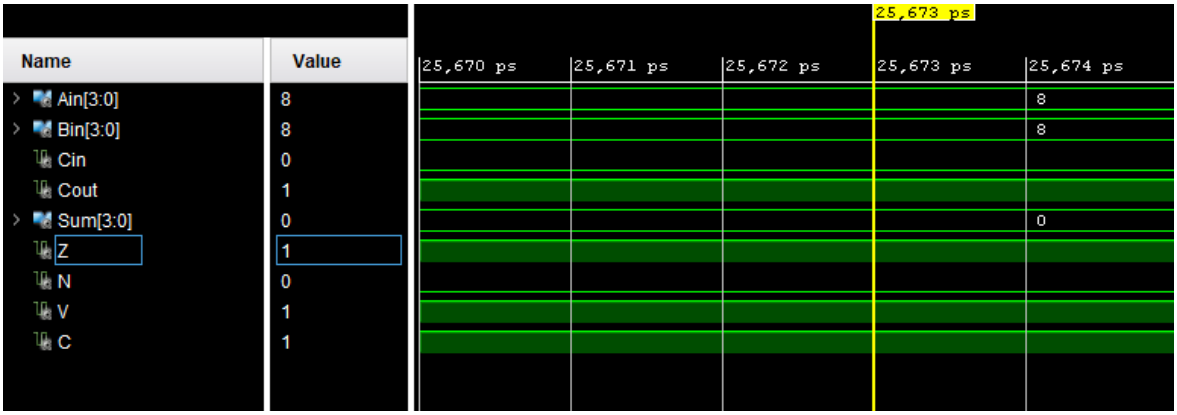
```
        Bin <= "1000";
        Cin <= '0';
        wait for 10 ns;
        Ain <= "0100";
        Bin <= "0011";
        Cin <= '0';
        wait for 10 ns;
        Ain <= "1111";
        Bin <= "0000";
        Cin <= '0';
        wait for 10 ns;
    -- subtraction
        Ain <= "0000";
        Bin <= "0000";
        Cin <= '1';
        wait for 10 ns;
        Ain <= "0001";
        Bin <= "0001";
        Cin <= '1';
        wait for 10 ns;
        Ain <= "1000";
        Bin <= "1000";
        Cin <= '1';
        wait for 10 ns;
        Ain <= "0100";
        Bin <= "0011";
        Cin <= '1';
        wait for 10 ns;
        Ain <= "1111";
        Bin <= "0000";
        Cin <= '1';
        wait for 10 ns;


    -- Put test bench stimulus code here

     wait;
   end process;


end;
```

3.7 Present the simulation results of the test bench to show that the added flags work as expected.

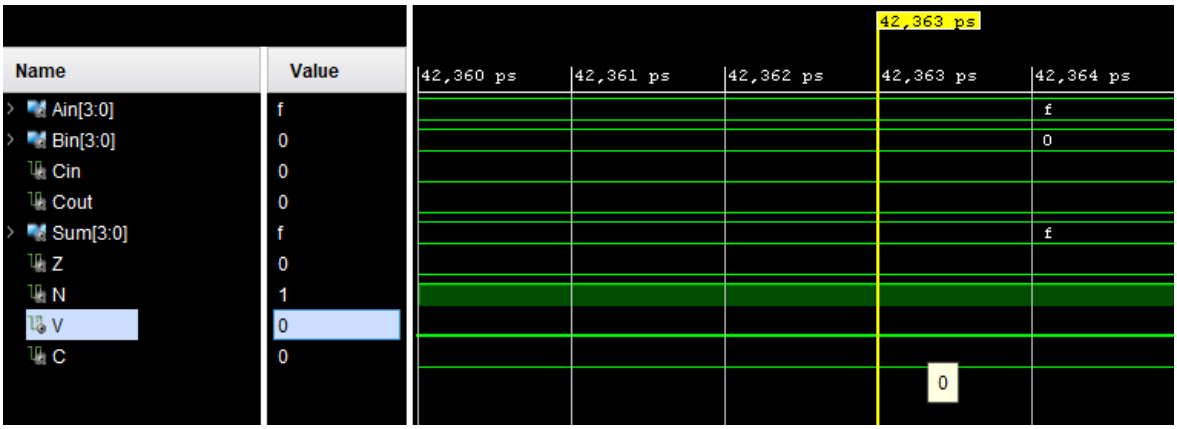
Z FLAG



Overflow V and carry flags



Negative flag



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.

   The lab took 8 hours to complete
2. Write a few sentences describing the purpose of this lab.

   The purpose was to design circuits(fulladder/subtractor) using VHDL and testing them using a test bench to confirm the functionality.

3. Did your full adder pass tests for all eight possible inputs?

   Yes, the full adder passed all the test and can be confirmed by referencing the results posted above.
4. How did you test your 4-bit adder/subtractor?

   This was tested by using a test-bench that first tested some basic addition and then tested some subtraction operations the results can be seen above.