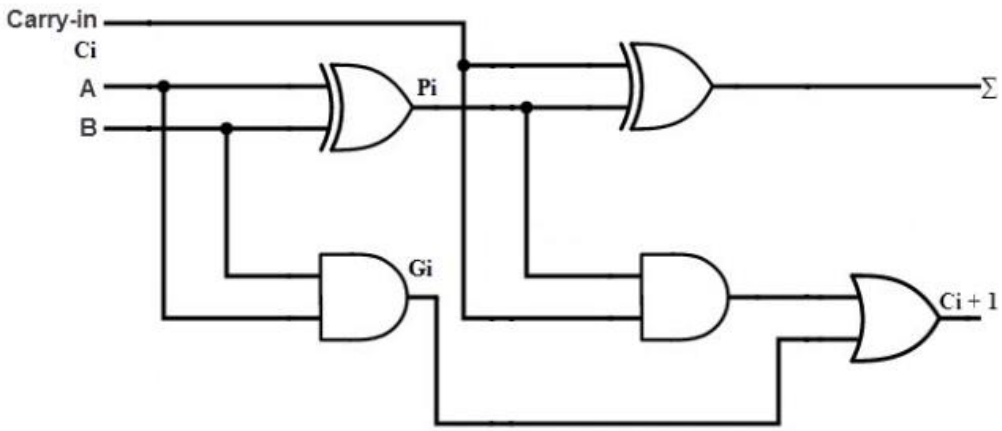
پارسا محمدپور – ۹۸۲۴۳۰۵۰

پویا جهانگیری – ۹۸۲۴۳۰۷۶

گزارش آزمایش جلسه گذشته:

در این مثال از ما خواسته شده بود تا کدهای VHDL مربوط به یک full-adder ای را بزنیم که قادر به انجام عملیات‌های جمع تک بیتی و همچنین تفریق تک بیتی باشد. البته منظور از عملیات تفریق، در اصل همان جمع‌کردن بیت اول و حاصل xor بیت دوم و ورودی op می‌باشد. برای پیاده‌سازی این مژول، ابتدا به مدار آن نگاهی می‌اندازیم. مدار تک بیتی این ماژول به صورت زیر می‌باشد:



همانطور که در شکل بالا مشخص است، بیت خروجی حاصل xor شدن a و b و cin می‌باشد. اما در رابطه با بیت carry-out، ما مطابق مدار بالا عمل نکردیم، چون در آن از گیت xor استفاده شده بود و با توجه به گفته صورت سوال، چون تاخییر گیت xor بیشتر از and و or بود، به جای این کار، از عبارت زیر استفاده کردیم:

Carry-out = (a and b) or ((a or b) and carry-in)

اما به سراغ دلیل انجام این کار می‌رویم. دلیل انجام این کار این بود که خواستیم مدت زمان پاسخ‌دهی سیستم را کمتر کنیم. در این حالت گفته شده در شکل بالا، زمان مورد نیز برای اینکه بیت carry-out را مشخص کنیم،، برابر با مقدار زیر است:

$$t_{carry-out} = t_{xor} + t_{and} + t_{or} = 10 + 5 + 5 = 20$$

اما مقدار تاخییر عبارت نوشته شده در بالا برابر با مقدار زیر است:

$$t_{carry-out} = t_{or} + t_{and} + t_{or} = 5 + 5 + 5 = 15$$

همانطور که در دیدیم، مقدار تاخییر بیت carry-out برای عبرت نوشته شده، کمتر از مقدار بدست آمده برای شکل بالا می‌باشد. به همین منظور، برای بیت carry-out، از عبارت نوشته‌شده استفاده کردیم.

حال به سراغ توضیح پیاده‌سازی‌های صورت گرفته می‌رویم. باتوجه به اینکه در سورت سوال از ما خواسته شده‌است تا تاخییر گیت‌های گفته شده را به میزان مشخصی قرار دهیم، پس دو تا کار می‌توانیم انجام‌دهیم:

۱- برای هر کدام از این دو خروجی، میزان تاخییر را محاسبه نموده و سپس آن را با استفاده از المان ها بنویسیم و سپس در انتها عبارت after t ns بگذاریم که در آن t میزان تاخییر می‌باشد.

۲- در این راه، می‌توانیم، ابتدا هرکدام از گیت‌های پایه داده شده در صورت سوال را به صورت یک ماژول پیاده‌سازی کنیم و مقدار تاخییر مورد نظر را با استفاده از عبارت after برای آن لحاظ کنیم، سپس در بقیه جاها به جای استفاده از گیت‌های عادی، از ماژول‌های خودمان استفاده کنیم که تاخییر در آن‌ها لحاظ شده است.

ما برای این آزمایش از حالت دوم استفاده کردیم و ابتدا گیت‌های پایه گفته شده در صورت سوال را به عنوان یک ماژول پیاده‌سازی کردیم و سپس از آن ماژول استفاده کردیم. کدهای این ماژول‌ها به صورت زیر می‌باشد:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity myOr is
 port(
    a,b : in std_logic;
    o: out std_logic
 );
end myOr;

architecture bhv of myOr is
begin

    o <= a or b after 5 ns;
end bhv;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity myNot is
 port(
    a,b : in std_logic;
    o: out std_logic
 );
end myNot;

architecture bhv of myNot is
begin

    o <= not a after 2 ns;
end bhv;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity myAnd is
 port(
    a,b : in std_logic;
    o: out std_logic
 );
end myAnd;

architecture bhv of myAnd is
begin

    o <= a and b after 5 ns;
end bhv;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity myXor is
 port(
    a,b : in std_logic;
    o: out std_logic
 );
end myXor;

architecture bhv of myXor is
begin

    o <= a xor b after 15 ns;
end bhv;
```

حال در ادامه، برای پیاده‌سازی full-adder از این ماژول‌ها استفاده خواهیم کرد.

در پیاده‌سازی از ماژول full-adder، طبق توضیحات ارائه شده در بالاتر عمل کردیم. کدهای مربوط به این قسمت به صورت زیر می‌باشد:

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3
4   entity fulladder is
5    port(
6       a,b,cin : in std_logic;
7       o, cout: out std_logic
8    );
9   end fulladder;
10
11  architecture bhv of fulladder is
12   -- Component Declaration for the Unit Under Test (UUT)
13
14   COMPONENT myAnd
15   PORT(
16   a, b : IN STD_LOGIC;
17   o : OUT STD_LOGIC
18   );
19   END COMPONENT;
20
21   COMPONENT myOr
22   PORT(
23   a, b : IN STD_LOGIC;
24   o : OUT STD_LOGIC
25   );
26   END COMPONENT;
27
28   COMPONENT myNot
29   PORT(
30   a : IN STD_LOGIC;
31   o : OUT STD_LOGIC
32   );
33   END COMPONENT;
34
35   COMPONENT myXor
36   PORT(
37   a, b : IN STD_LOGIC;
38   o : OUT STD_LOGIC
39   );
40   END COMPONENT;
41
```

```
41
42      --Inputs
43      signal a_xor_b : STD_LOGIC;
44      signal a_and_b : STD_LOGIC;
45      signal a_or_b : std_logic;
46      signal cin_and_a_or_b : STD_LOGIC;
47      signal o_res : STD_LOGIC;
48      signal cout_res : STD_LOGIC;
49
50  begin
51      -- Instantiating
52      a_xor_b_instance: myXor PORT MAP (
53          a => a,
54          b => b,
55          o => a_xor_b
56      );
57      a_and_b_instance: myAnd PORT MAP(
58          a => a,
59          b => b,
60          o => a_and_b
61      );
62      a_or_b_instance: myOr port MAP(
63          a => a,
64          b => b,
65          o => a_or_b
66      );
67      cin_and_a_or_b_instance: myAnd PORT MAP(
68          a => cin,
69          b => a_or_b,
70          o => cin_and_a_or_b
71      );
72      o_res_instance: myXor port MAP(
73          a => a_xor_b,
74          b => cin,
75          o => o_res
76      );
77      cout_res_instance: myOr port MAP(
78          a => a_and_b,
79          b => cin_and_a_or_b,
80          o => cout_res
```

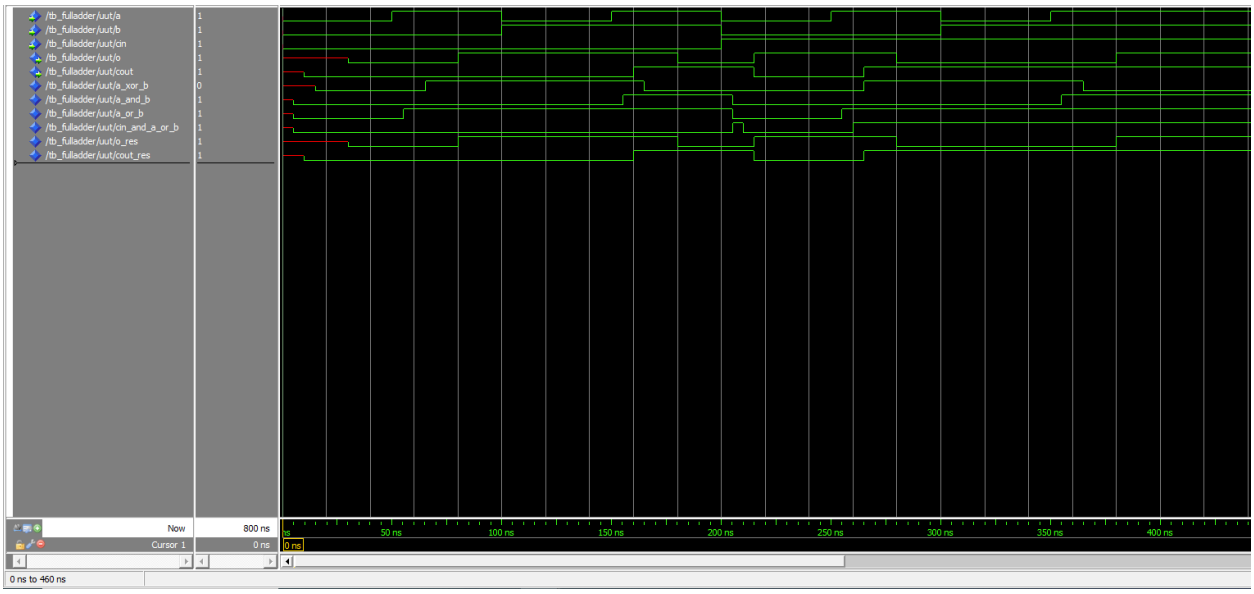
```
66      );
67      cin_and_a_or_b_instance: myAnd PORT MAP(
68          a => cin,
69          b => a_or_b,
70          o => cin_and_a_or_b
71      );
72      o_res_instance: myXor port MAP(
73          a => a_xor_b,
74          b => cin,
75          o => o_res
76      );
77      cout_res_instance: myOr port MAP(
78          a => a_and_b,
79          b => cin_and_a_or_b,
80          o => cout_res
81      );
82
83      o <= o_res;
84      cout <= cout_res;
85
86  end bhv;
```

حال برای تست کردن کدهای زده شده، یک فایل تست‌بنچ می‌نویسیم و ورودی‌های مختلف ر به آن می‌دهیم و خروجی را مشاهده می‌کنیم.

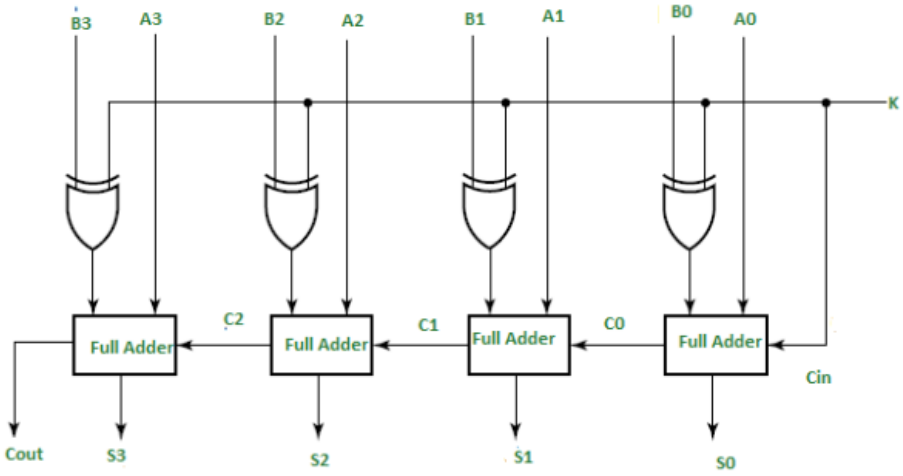
فایل تست‌بنچ برای full-adder به صورت زیر می‌باشد:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_fulladder IS
END tb_fulladder;

ARCHITECTURE behavior OF tb_fulladder IS

 -- Component Declaration for the Unit Under Test (UUT)

 COMPONENT fulladder
 PORT(
 a,b,cin : in std_logic;
 o, cout: out std_logic
 );
 END COMPONENT;

 --Inputs
 signal a, b, cin : std_logic;

 --Outputs
 signal o, cout : std_logic;
 -- appropriate port name

BEGIN

 -- Instantiate the Unit Under Test (UUT)
 uut: fulladder PORT MAP (
 a => a,
 b => b,
 cin => cin,
 o => o,
 cout => cout
 );

 -- Stimulus process
 stim_proc: process
 begin

 stim_proc: process
 begin

 a <= '0';
 b <= '0';
 cin <='0';
 wait for 50 ns;
 a <= '1';
 b <= '0';
 cin <='0';
 wait for 50 ns;
 a <= '0';
 b <= '1';
 cin <='0';
 wait for 50 ns;
 a <= '1';
 b <= '1';
 cin <='0';
 wait for 50 ns;
 a <= '0';
 b <= '0';
 cin <='1';
 wait for 50 ns;
 a <= '1';
 b <= '0';
 cin <='1';
 wait for 50 ns;
 a <= '0';
 b <= '1';
 cin <='1';
 wait for 50 ns;
 a <= '1';
 b <= '1';
 cin <='1';
 wait for 50 ns;

 wait;
 end process;

END;
```

حال شبیه‌سازی را در نرم‌افزار مدل‌سیم انجام می‌دهیم. عکس‌های حاصل از این شبیه‌سازی به صورت زیر می‌باشد:



حال به سراغ پیاده‌سازی ماژول adder-subtracter چهار بیتی می‌رویم. برای این کار مشبه با تصویر زیر عمل می‌کنیم:



مشابه با تصویر بالا عمل می‌کنیم و پیاده‌سازی را انجام می‌دهیم. کد پیاده‌سازی شده به صورت زیر می‌باشد:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity addersubtracter is
 port(
     sel : in STD_LOGIC;
     a,b : in std_logic_VECTOR(3 downto 0);
     res: out std_logic_VECTOR(3 downto 0);
     overflow, cout : OUT std_logic

 );
end addersubtracter;

architecture bhv of addersubtracter is
  -- Component Declaration for the Unit Under Test (UUT)

  COMPONENT myAnd
  PORT(
  a, b : IN STD_LOGIC;
  o : OUT STD_LOGIC
  );
  END COMPONENT;

  COMPONENT myOr
  PORT(
  a, b : IN STD_LOGIC;
  o : OUT STD_LOGIC
  );
  END COMPONENT;

  COMPONENT myNot
  PORT(
  a : IN STD_LOGIC;
  o : OUT STD_LOGIC
  );
  END COMPONENT;

  COMPONENT myXor
  PORT(
  a, b : IN STD_LOGIC;
  o : OUT STD_LOGIC
```

```
  a, b : IN STD_LOGIC;
  o : OUT STD_LOGIC
  );
  END COMPONENT;

  COMPONENT fulladder
  PORT(
     a,b,cin : in std_logic;
     o, cout: out std_logic
  );
  END COMPONENT;

  --Inputs
  signal b_xor_sel, res_signal : STD_LOGIC_VECTOR(3 downto 0);
  signal c1, c2, c3, c4, overflow_res : STD_LOGIC;

begin
     -- Instantiating
     b_xor_sel_instance0: myXor PORT MAP (
         a => b(0),
         b => sel,
         o => b_xor_sel(0)
     );
     b_xor_sel_instance1: myXor PORT MAP (
         a => b(1),
         b => sel,
         o => b_xor_sel(1)
     );
     b_xor_sel_instanc2: myXor PORT MAP (
         a => b(2),
         b => sel,
         o => b_xor_sel(2)
     );
     b_xor_sel_instance3: myXor PORT MAP (
         a => b(3),
         b => sel,
         o => b_xor_sel(3)
     );

     full_adder_instance0: fulladder PORT MAP(
         a => a(0),
```

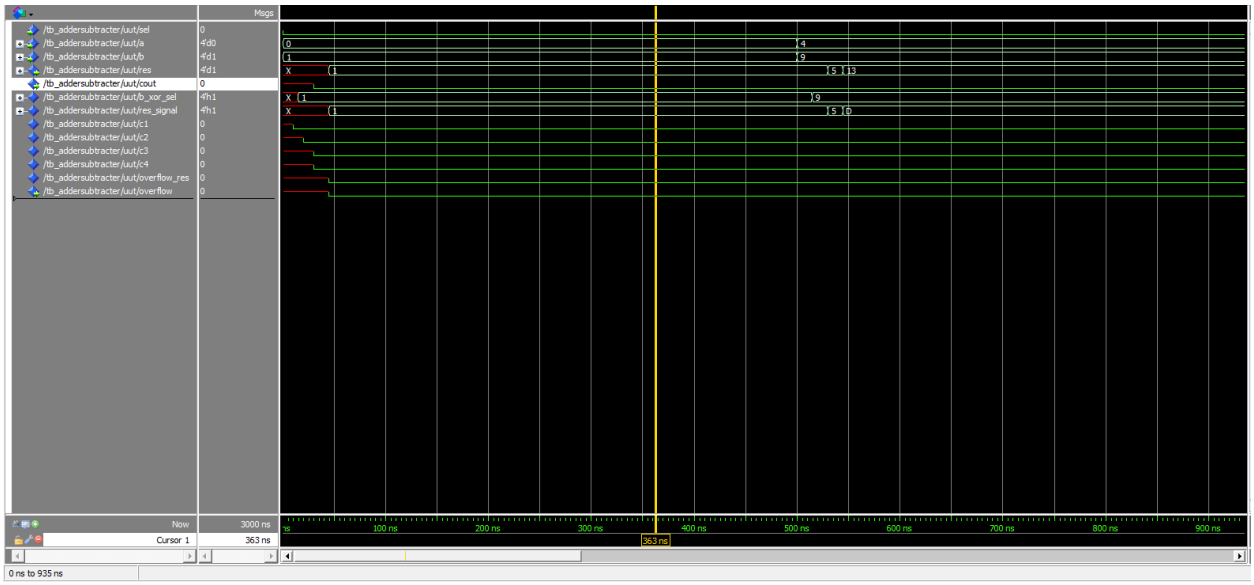
```
    full_adder_instance0: fulladder PORT MAP(
        a => a(0),
        b => b_xor_sel(0),
        cin => sel,
        o => res_signal(0),
        cout => c1
    );
    full_adder_instance1: fulladder PORT MAP(
        a => a(1),
        b => b_xor_sel(1),
        cin => c1,
        o => res_signal(1),
        cout => c2
    );
    full_adder_instance2: fulladder PORT MAP(
        a => a(2),
        b => b_xor_sel(2),
        cin => c2,
        o => res_signal(2),
        cout => c3
    );
    full_adder_instance3: fulladder PORT MAP(
        a => a(3),
        b => b_xor_sel(3),
        cin => c3,
        o => res_signal(3),
        cout => c4
    );

    overflow_instance: myXor PORT MAP(
        a => c3,
        b => c4,
        o => overflow_res
    );

    overflow <= overflow_res;
    res <= res_signal;
    cout <= c4;

end bhv;
```

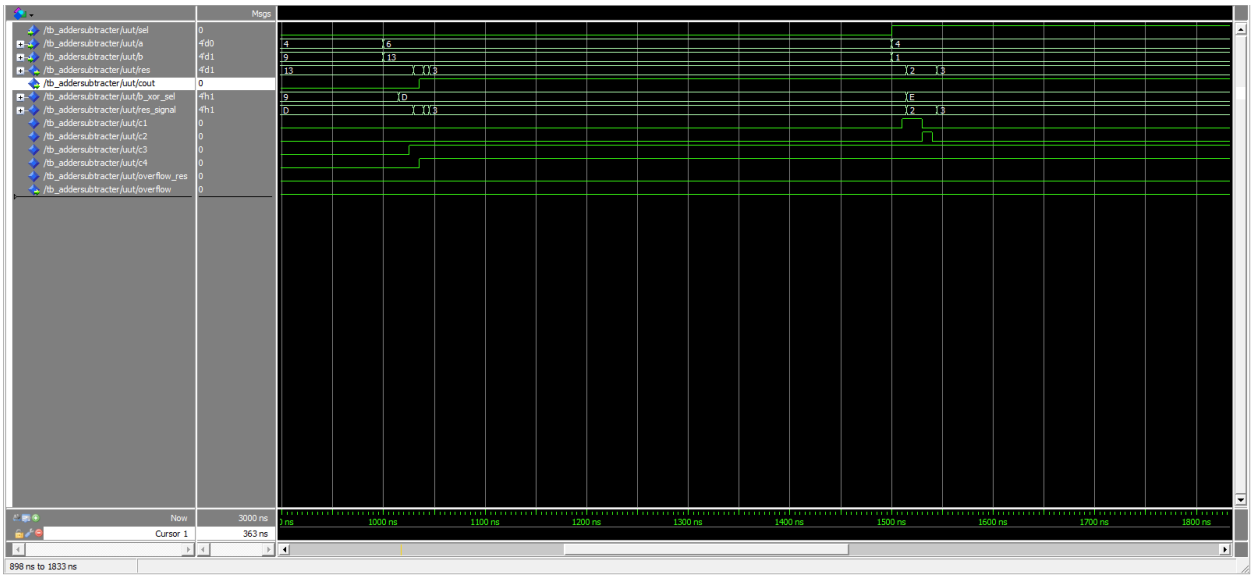
حال برای اینکه از صحت عملکرد این مدار مطمئن شویم، یک فایل تست‌بنچ برای آن می‌نویسیم و ورودی‌های مختلفی را به آن اعمال

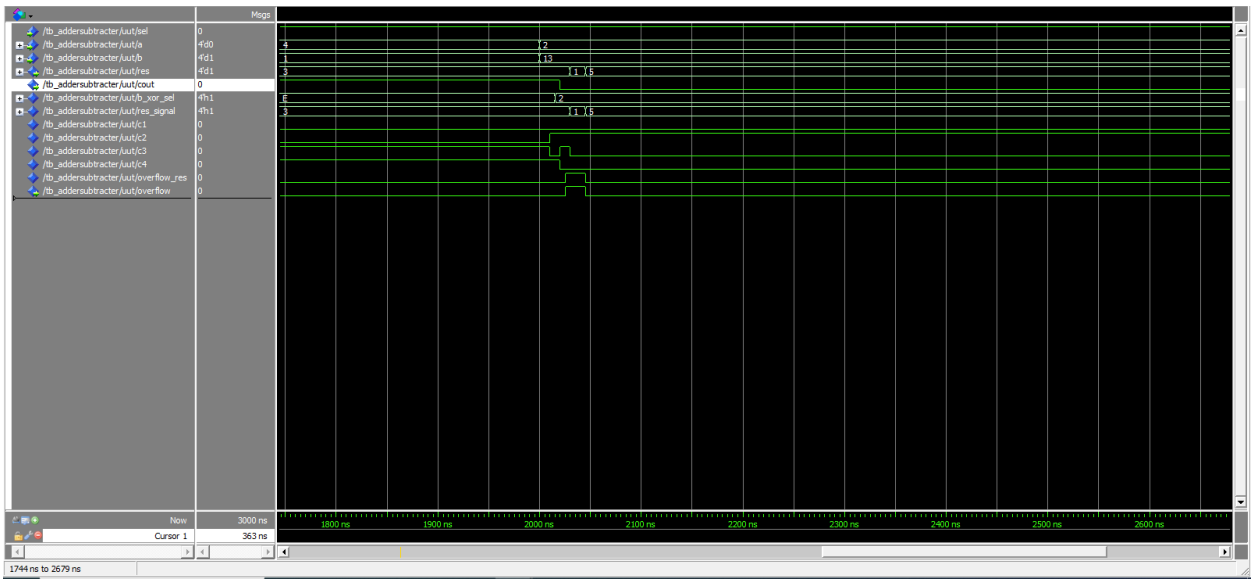
می‌کنیم و سپس خروجی را بررسی می‌کنیم. کدهای مربوط به فایل تست‌بنچ به صورت زیر می‌باشد:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_addersubtracter IS
END tb_addersubtracter;

ARCHITECTURE behavior OF tb_addersubtracter IS

 -- Component Declaration for the Unit Under Test (UUT)

 COMPONENT addersubtracter
 PORT(
    sel : in STD_LOGIC;
    a,b : in std_logic_VECTOR(3 downto 0);
    res: out std_logic_VECTOR(3 downto 0);
    overflow, cout : OUT std_logic
 );
 END COMPONENT;

 --Inputs
 signal a, b, res : std_logic_VECTOR(3 downto 0);
 signal sel : std_logic;

 --Outputs
 signal overflow, cout : std_logic;
 -- appropriate port name

BEGIN

 -- Instantiate the Unit Under Test (UUT)
 uut: addersubtracter PORT MAP (
 sel => sel,
 a => a,
 b => b,
 res => res,
 overflow => overflow,
 cout => cout
 );

 -- Stimulus process
 stim_proc: process
 -- Stimulus process
 stim_proc: process
 begin

 a <= "0000";
 b <= "0001";
 sel <= '0'; -- 0 is plus and 1 is subtract
 wait for 500 ns;
 a <= "0100";
 b <= "1001";
 sel <= '0'; -- 0 is plus and 1 is subtract
 wait for 500 ns;
 a <= "0110";
 b <= "1101";
 sel <= '0'; -- 0 is plus and 1 is subtract
 wait for 500 ns;
 a <= "0100";
 b <= "0001";
 sel <= '1'; -- 0 is plus and 1 is subtract
 wait for 500 ns;
 a <= "0010";
 b <= "1101";
 sel <= '1'; -- 0 is plus and 1 is subtract
 wait for 500 ns;

 wait;
 end process;

END;
```

حال به سراغ شبیه‌سازی فایل تست‌بنچ می‌رویم. خروجی حاصل از شبیه‌سازی به صورت زیر می‌باشد:







حال همانطور که در تصویر می‌بینیم، تنها خروجی که عجیب بود، خروجی‌ای بود که در آن حاصل منفی شده است، آن هم ب این دلیل است

که نمایش اعداد منفی کلا متفاوت است و باید **مکمل‌دو** باشد.

پیش‌گزارش آزمایش بعدی:

در کلاس کدهای مربوط به آزمایش بعدی به طور کامل زده شد و به استاد تحویل داده‌شد. عکس فایل پیاده‌سازی شده به صورت زیر می‌باشد:

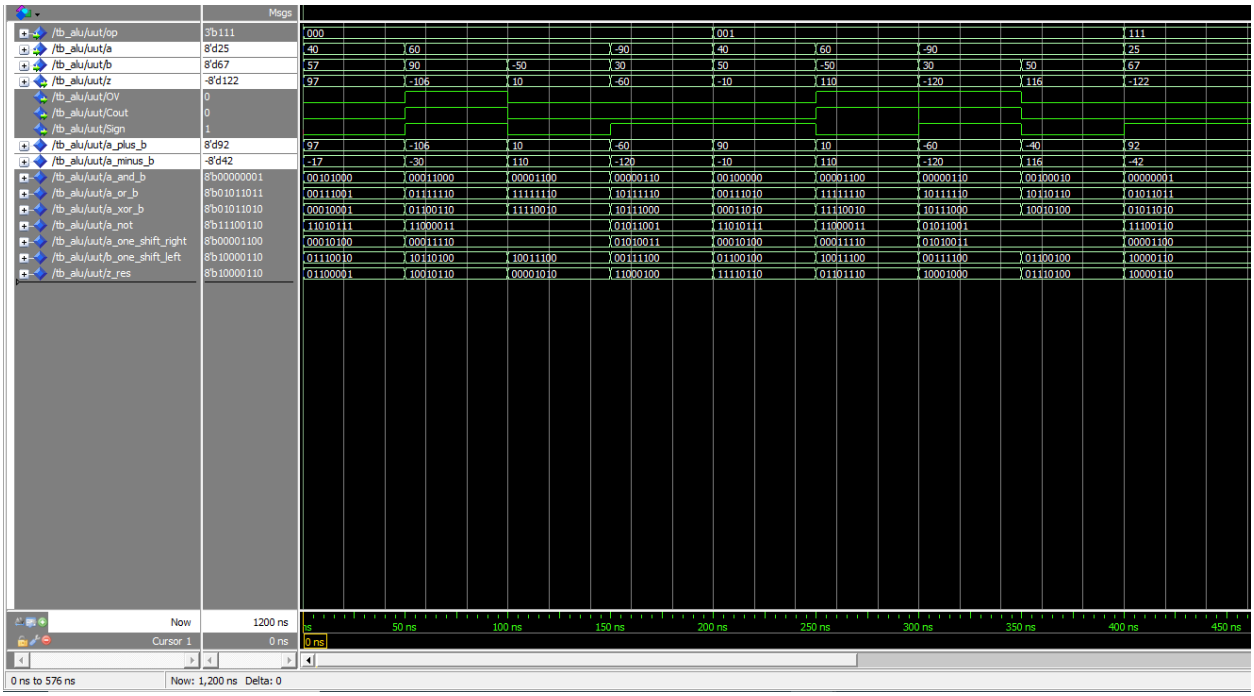
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity alu is
 port(
    op : in STD_LOGIC_VECTOR(2 downto 0);
    a,b : in SIGNED(7 downto 0);
    z: out SIGNED(7 downto 0);
    OV, Cout, Sign : OUT std_logic
 );
end alu;

architecture bhv of alu is
 -- Component Declaration for the Unit Under Test (UUT)

 signal a_plus_b, a_minus_b, a_and_b, a_or_b, a_xor_b, a_not, a_one_shift_right, b_one_shift_left, z_res : SIGNED(7 downto 0);

begin
    a_plus_b <= a + b;
    a_minus_b <= a - b;
    a_or_b <= a or b;
    a_xor_b <= a xor b;
    a_and_b <= a and b;
    a_not <= not a;
    a_one_shift_right <= a srl 1;
    b_one_shift_left <= b sll 1;
```

```
begin
    a_plus_b <= a + b;
    a_minus_b <= a - b;
    a_or_b <= a or b;
    a_xor_b <= a xor b;
    a_and_b <= a and b;
    a_not <= not a;
    a_one_shift_right <= a srl 1;
    b_one_shift_left <= b sll 1;

    z_res <= a_plus_b when op = "000" else
        a_minus_b when op = "001" else
        a_and_b when op = "010" else
        a_or_b when op = "011" else
        a_xor_b when op = "100" else
        a_not when op = "101" else
        a_one_shift_right when op = "110" else
        b_one_shift_left when op = "111";

    Cout <=  ((a(7) and b(7) and (not z_res(7))) or ((not a(7)) and (not b(7)) and z_res(7))) when op = "000" else
          ((a(7) and (not b(7)) and z_res(7)) or ((not a(7)) and b(7) and (not z_res(7))))when op = "001" else '0';
    z <= z_res(7 downto 0);
    OV <= ((a(7) and b(7) and (not z_res(7))) or ((not a(7)) and (not b(7)) and z_res(7))) when op = "000" else
          ((a(7) and (not b(7)) and z_res(7)) or ((not a(7)) and b(7) and (not z_res(7))))when op = "001" else '0';

    Sign <= z_res(7);

end bhv;
```

حال در ادامه کدهای فایل مربوط به تست‌بنچ را قرار می‌دهیم:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

ENTITY tb_alu IS
END tb_alu;

ARCHITECTURE behavior OF tb_alu IS

 -- Component Declaration for the Unit Under Test

 COMPONENT alu
 PORT(
    op : in STD_LOGIC_VECTOR(2 downto 0);
    a,b : in SIGNED(7 downto 0);
    z: out SIGNED(7 downto 0);
    OV, Cout, Sign : OUT std_logic
 );
 END COMPONENT;

 --Inputs & Outputs
 signal a, b, z : SIGNED(7 downto 0);
 signal op : STD_LOGIC_VECTOR(2 downto 0);
 signal OV, Cout, Sign : STD_LOGIC;
 -- appropriate port name

BEGIN

 -- Instantiate the Unit Under Test (UUT)
 uut: alu PORT MAP (
 op => op,
 a => a,
 b => b,
 z => z,
 OV => OV,
 Cout => Cout,
 Sign => Sign
 );

 -- Stimulus process
 stim_proc: process
 begin

 -- + Operation
 a <= to_signed(40, 8);
 b <= to_signed(57, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(60, 8);
 b <= to_signed(90, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(60, 8);
 b <= to_signed(-50, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(30, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(60, 8);
 b <= to_signed(-50, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(30, 8);
 op <= "000";
 wait for 50 ns;

 -- - Operation
 a <= to_signed(40, 8);
 b <= to_signed(50, 8);
 op <= "001";
 wait for 50 ns;
 a <= to_signed(60, 8);
 b <= to_signed(-50, 8);
 op <= "001";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(30, 8);
 op <= "001";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(50, 8);
 op <= "001";
 wait for 50 ns;


 -- & operation
 a <= to_signed(25, 8);
 b <= to_signed(67, 8);
 op <= "010";

 -- | operation
 op <= "011";

 -- XOR operaion
 op <= "100";

 -- ~ operation
 op <= "101";

 -- >> operation
 op <= "110";

 -- << operation
 op <= "111";

 wait;
 end process;

END;
```

در ادامه نتیجه حاصل از شبیه‌سازی را قرار می‌دهیم:



آموخته‌های این جلسه:

در این جلسه آموختیم که (مروری کردیم که) alu چگونه کار می‌کند. همچنین overflow کردن و underflow کردن را نیز آموختیم و یادآوری کردیم و همچنین مدار آن را یادآوری کردیم. همچنین تبدیل اعداد صحیح به علامت‌دار و بی‌علامت و بیت‌وکتور و ... را یادآوری کردیم. همچنین نحوه عملکرد SELECT... WHEN را هم مرور کردیم.