به نام خدا

پروژه پایانی آزمایشگاه مدارهای منطقی

استاد درس: مهندس دانش صفت دوست

اعضای گروه :

آنوشا شریعتی 9923041

فاطمه مولادوست 9923078

غزاله قاسم زاده 9823124

توضیحات مربوط به flag ها :

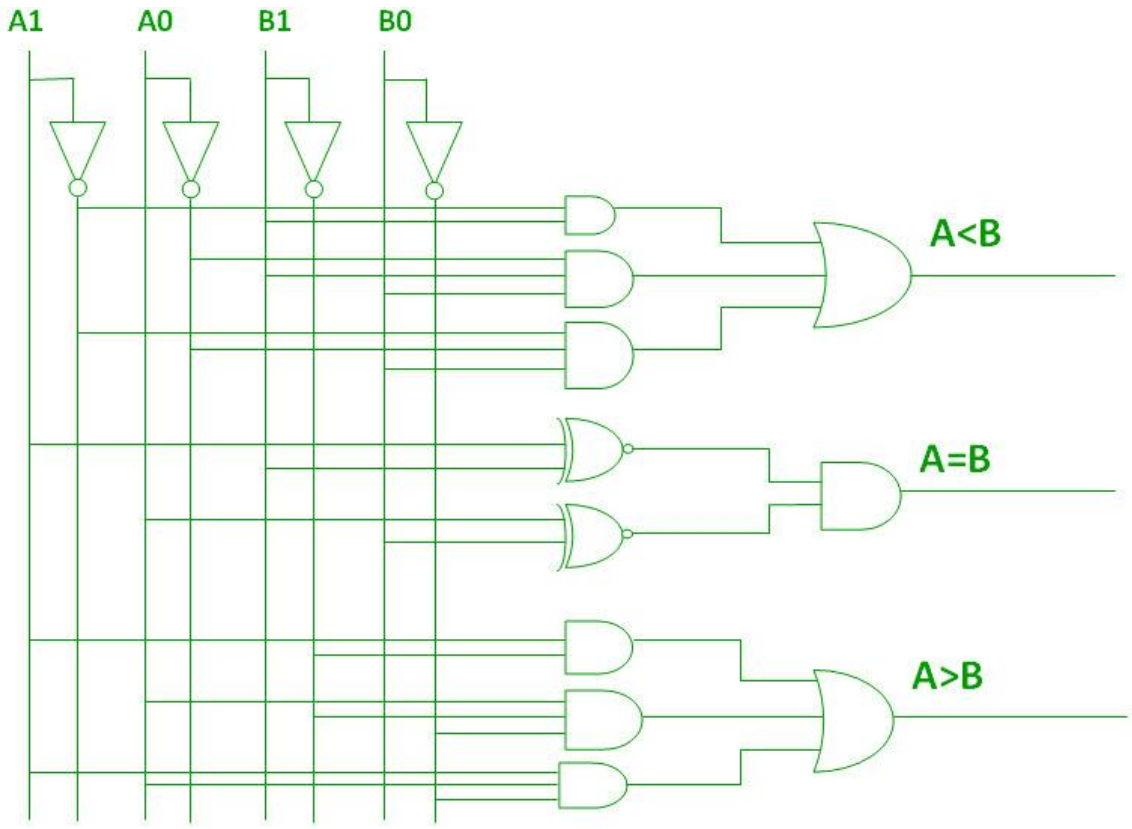
carry & overflow: این حالات هرگز رخ نمیدهند به همین دلیل مقدار صفر به flag آنها اختصاص داده شده است. در جمع دو عدد دوبیتی ماکسیمم تعداد بیت ها سه بیت است و در ضرب دو عدد دوبیتی ماکسیمم تعداد بیت ها چهار بیت میشود. خروجی ما چهاربیتی در نظر گرفته شده است که سرریز نداشته باشیم.

Negative (n): چون اعداد unsigned در نظر گرفته شده اند، فقط در تفریق این حالت رخ میدهد که در کد مربوطه (substraction) این شرط در نظر گرفته شده است.(اگر b>a شود آنگاه flag آن یک میشود.)

Zero (z): در تفریق کننده اگر a و b برابر باشند حاصل صفر میشود که این شرط نیز در نظر گرفته شده است. همچنین در ضرب کننده وقتی یکی از a یا b صفر باشد خروجی 0 میشود و در جمع کننده وقتی هر دو ورودی صفر باشند این flag 1 میشود.

- برای در نظر گرفتن شرط یک شدن flag های z و n مدار منطقی مربوط به مقایسه کننده دوبیتی را به دست آورده سپس با توجه به آن کدنویسی انجام شده است.

Logic circuit of 2 bit comparator:



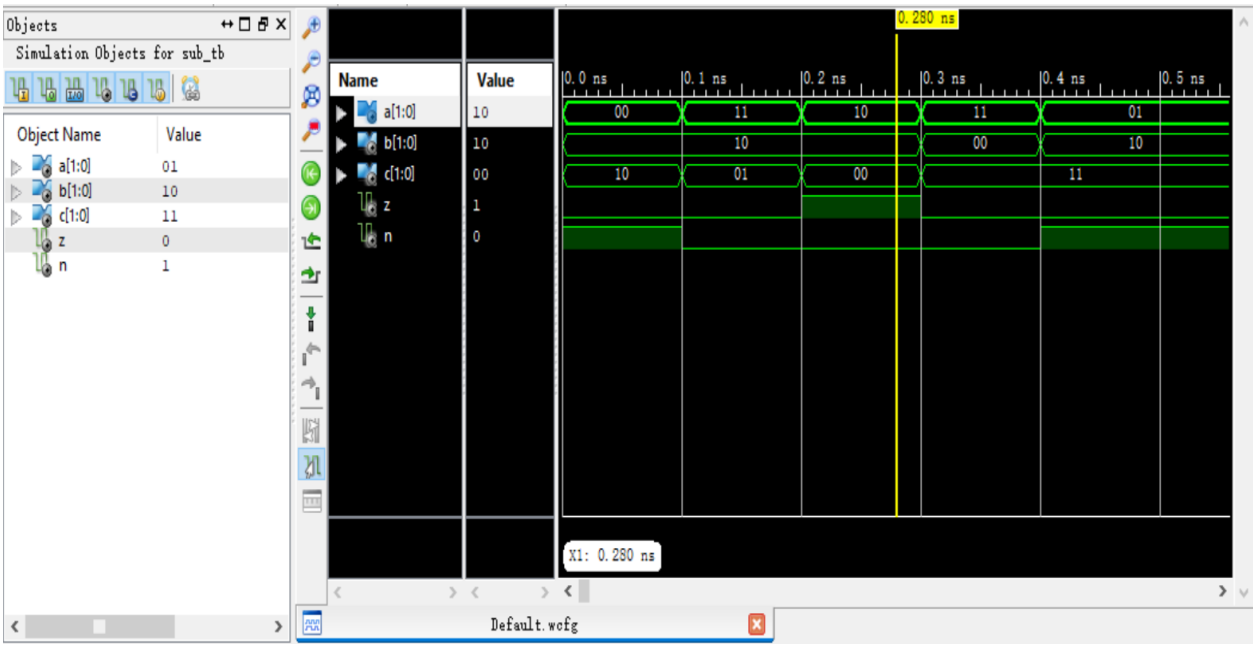
Substractor

Code:

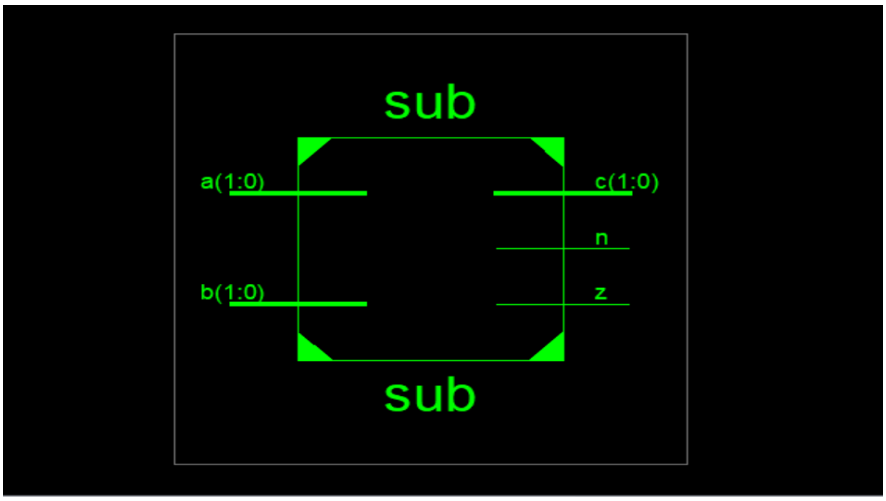
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity sub is
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           b : in  STD_LOGIC_VECTOR (1 downto 0);
           c : out  STD_LOGIC_VECTOR (1 downto 0);
           z : out  STD_LOGIC;
           n : out  STD_LOGIC);
end sub;

architecture Behavioral of sub is
 begin
 --substraction
 c <= STD_LOGIC_VECTOR(unsigned(a) - unsigned(b));

 --checking the flags
z <= not( (a(0) xor b(0)) or (a(1) xor b(1)));    --checks if a==b
n <= ((not a(1)) and b(1)) or ((not a(0)) and b(1) and b(0)) or ((not a(0)) and (not a(1)) and b(0));--checks if b>a


end Behavioral;

```

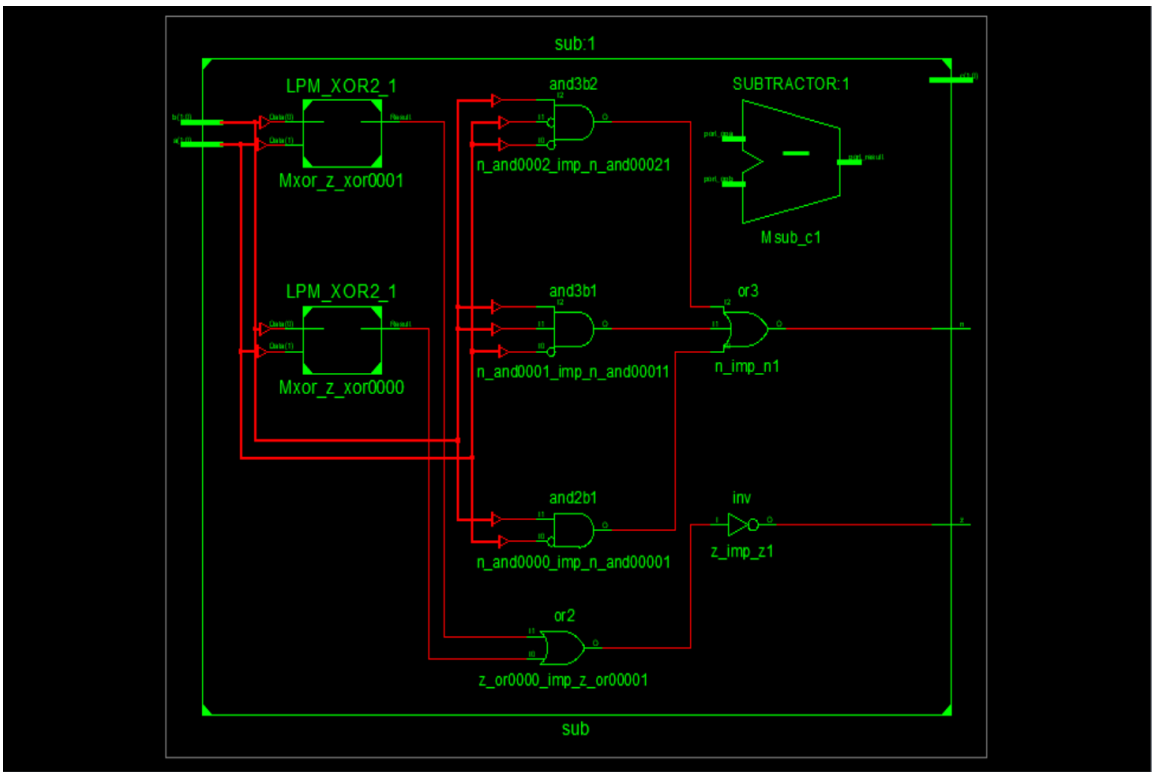
Testbench:

```
33          );
34
35
36      stim_proc: process
37      begin
38
39      a<="00";
40      b<="10";
41      wait for 100 ps;
42      a<="11";
43      b<="10";
44      wait for 100 ps;
45      a<="10";
46      b<="10";
47      wait for 100 ps;
48      a<="11";
49      b<="00";
50      wait for 100 ps;
51      a<="01";
52      b<="10";
53      wait for 100 ps;
54
55
56         wait;
57      end process;
```

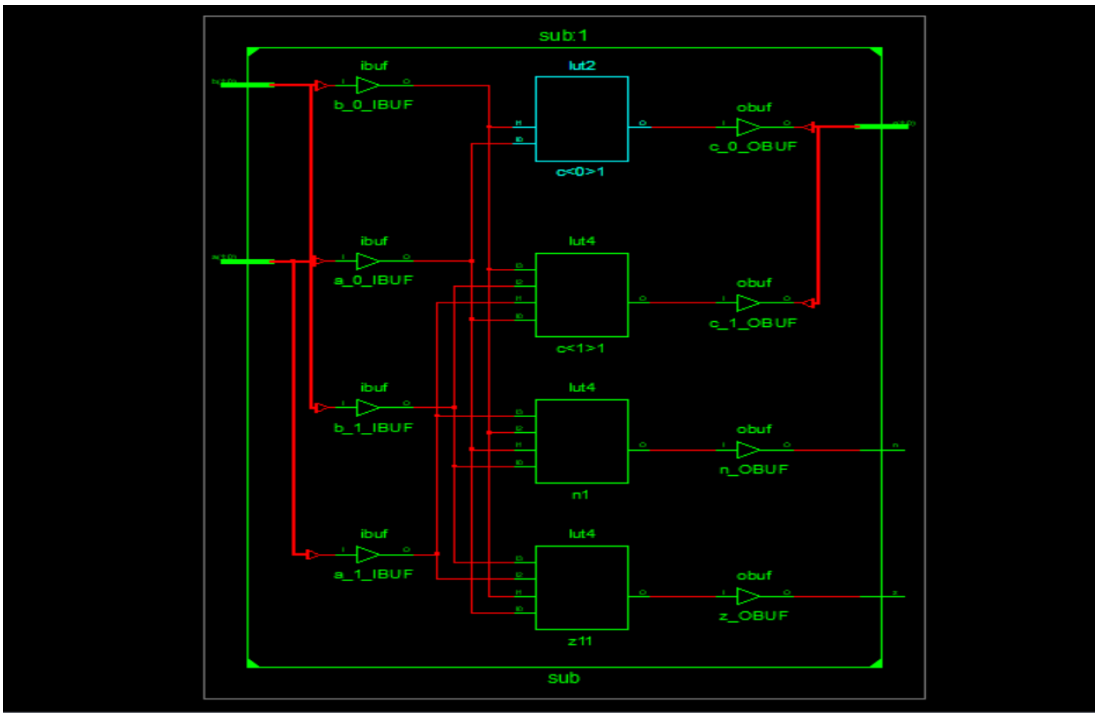


RTL Schematic:





Technology Schematic:

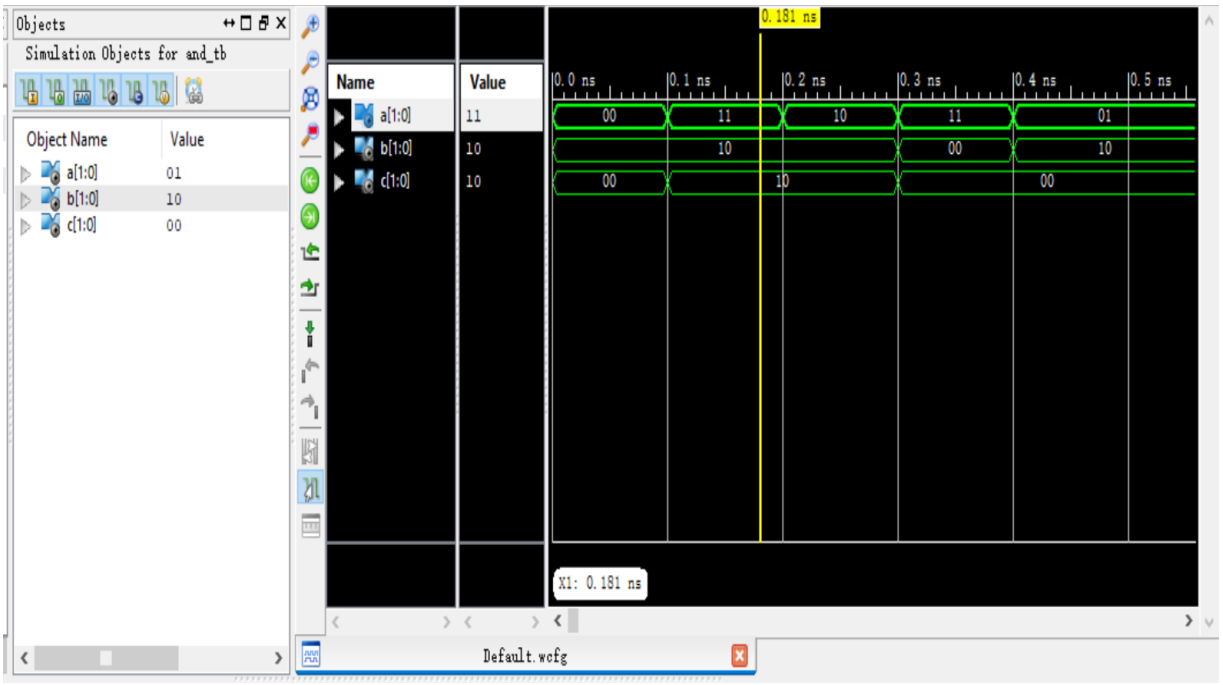


And:

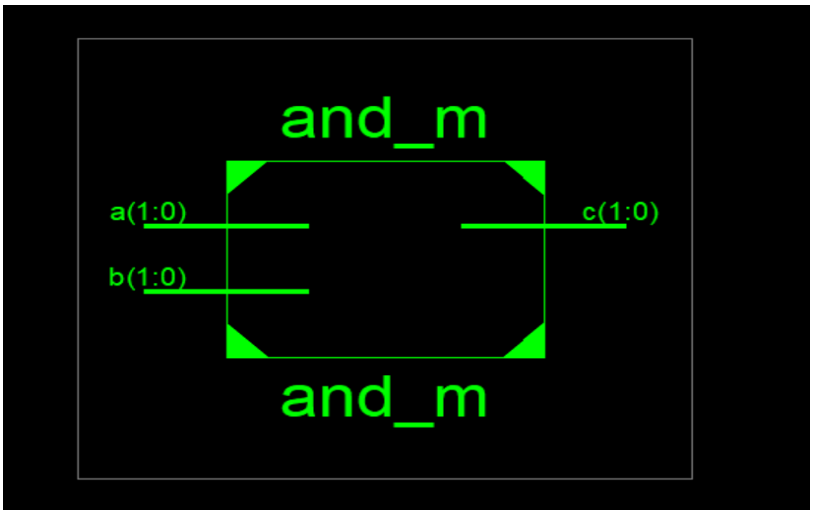
Code:

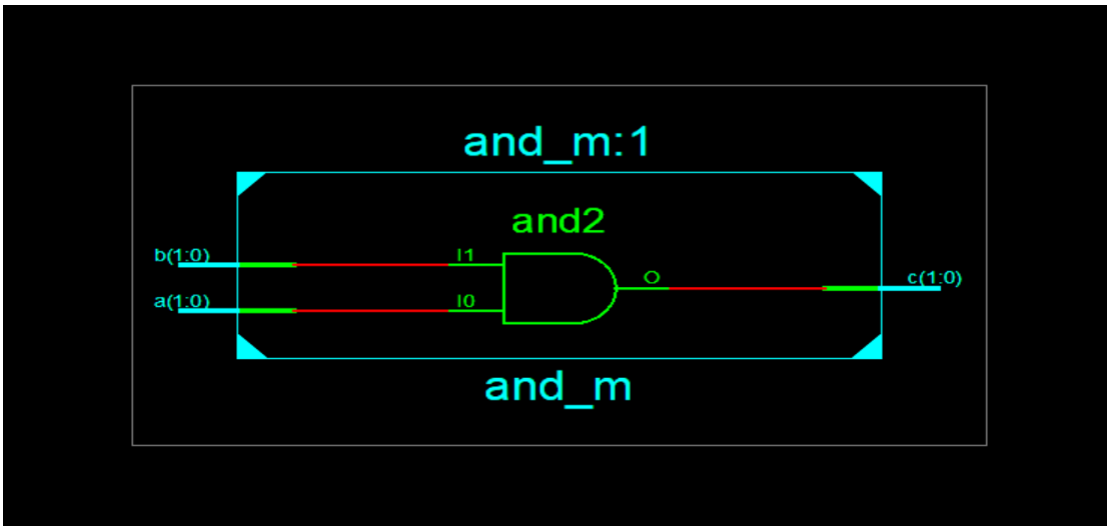
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity and_m is
5      Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
6             b : in  STD_LOGIC_VECTOR (1 downto 0);
7             c : out  STD_LOGIC_VECTOR (1 downto 0));
8  end and_m;
9
10 architecture Behavioral of and_m is
11
12 begin
13
14 c <= a and b ;
15
16 end Behavioral;
17
18
```

Testbench:

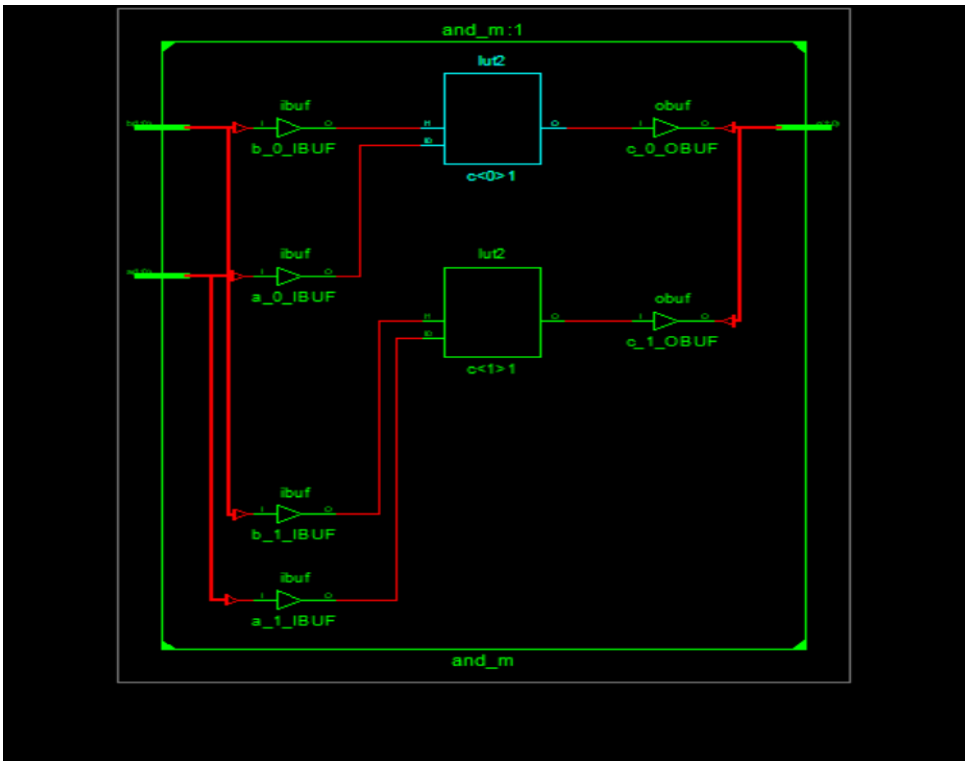


RTL schematic:





Technology Schematic:



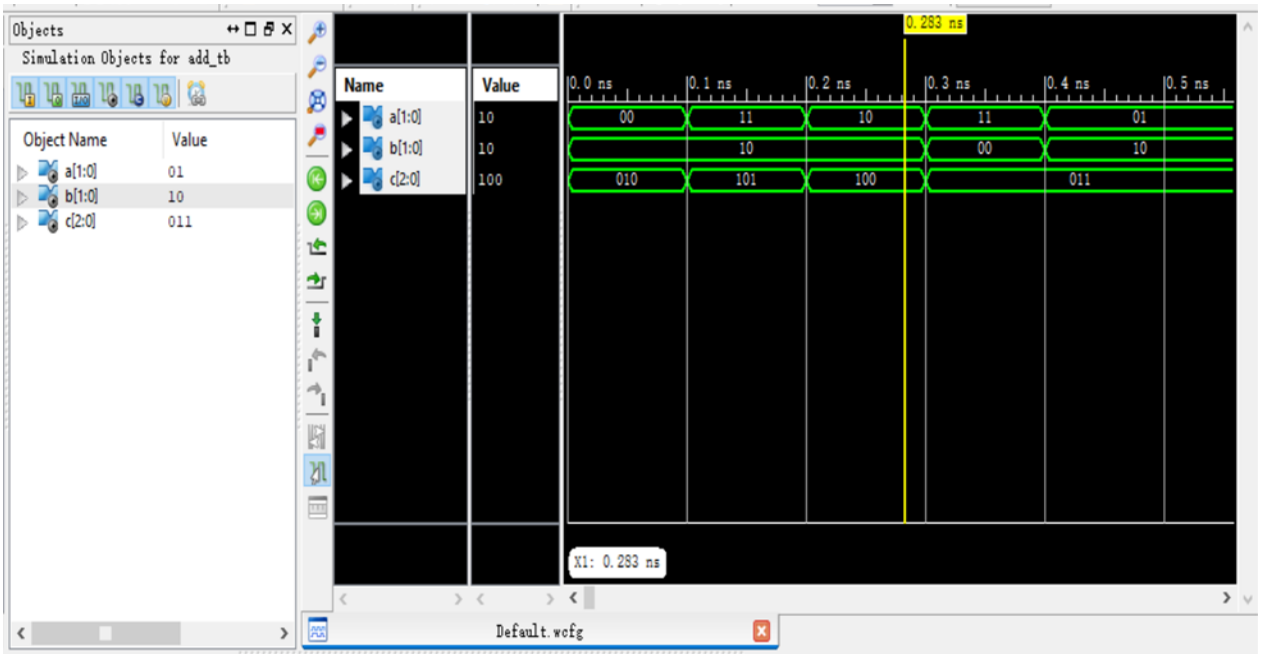
Adding:

همانطور که در بخش های قبل بیان شد طراحی به گونه ای انجام شده است که خروجی بزرگترین حالت در نظر گرفته شود(4 بیتی). به این دلیل زمانیکه حاصل عبارت ما دو بیتی باشد با هشدار مواجه میشویم و تست بنچ در بعضی حالات نتیجه درستی را ارائه نمیدهد پس باید همه ی اعداد 3 بیتی شوند در نتیجه برای رفع این مشکل در کد زیر یک صفر به اول اعداد اضافه میکنیم که اگر خروجی دوبیتی شد مشکلی در نمایش اعداد به وجود نیاید.

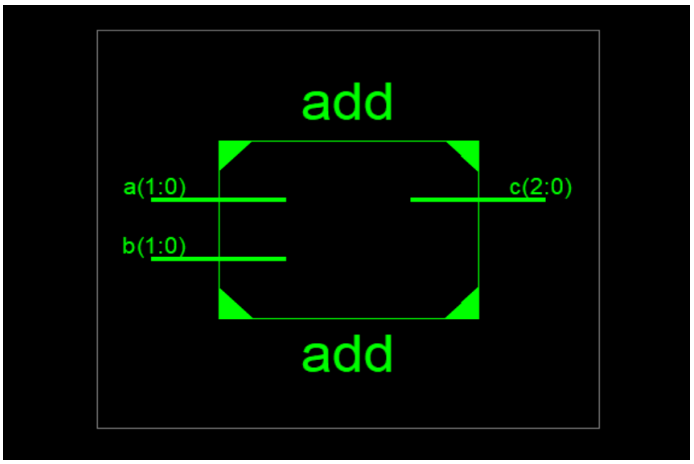
Code:

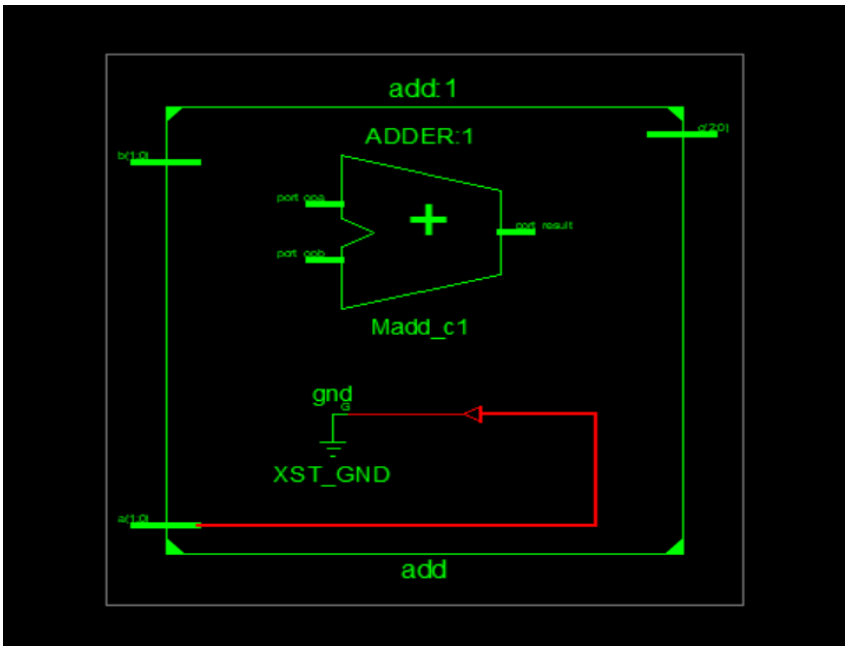
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  entity add is
6      Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
7             b : in  STD_LOGIC_VECTOR (1 downto 0);
8             c : out  STD_LOGIC_VECTOR (2 downto 0)); --the output has n+1 bits to prevent overflow
9  end add;
10
11 architecture Behavioral of add is
12
13 begin
14
15 c <=  STD_LOGIC_VECTOR(unsigned('0' & a) + unsigned('0' & b));
16 --adding a 0 to the begining of inputs in order to have two 3 bit numbers and avoid the warning
17
18 end Behavioral;
19
20
```

Testbench:

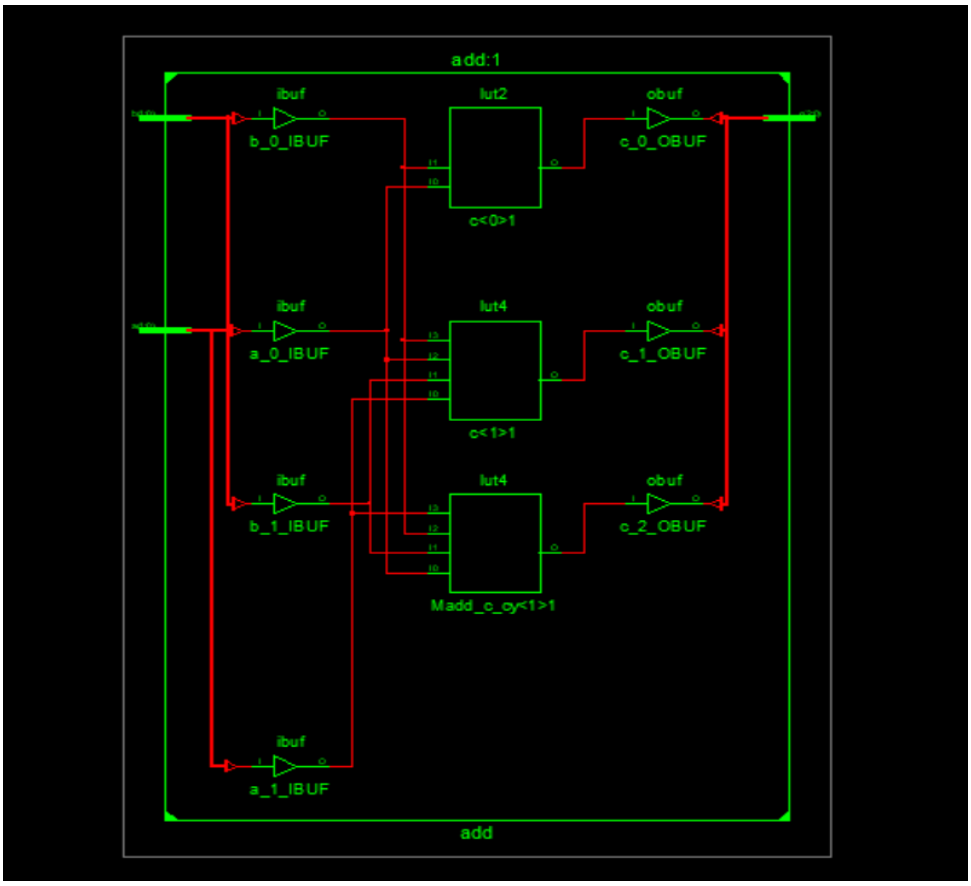


RTL Schematic:





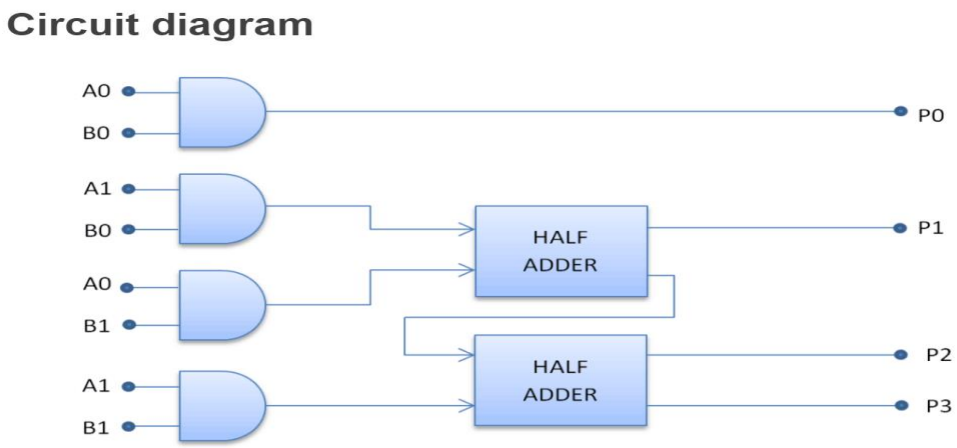
Technology Schematic:



Multiplier:

ابتدا شکل مدار منطقی ضرب کننده دو بیتی را به دست اورده و سپس بر اساس آن کد نویسی را انجام داده ایم.

Circuit diagram:

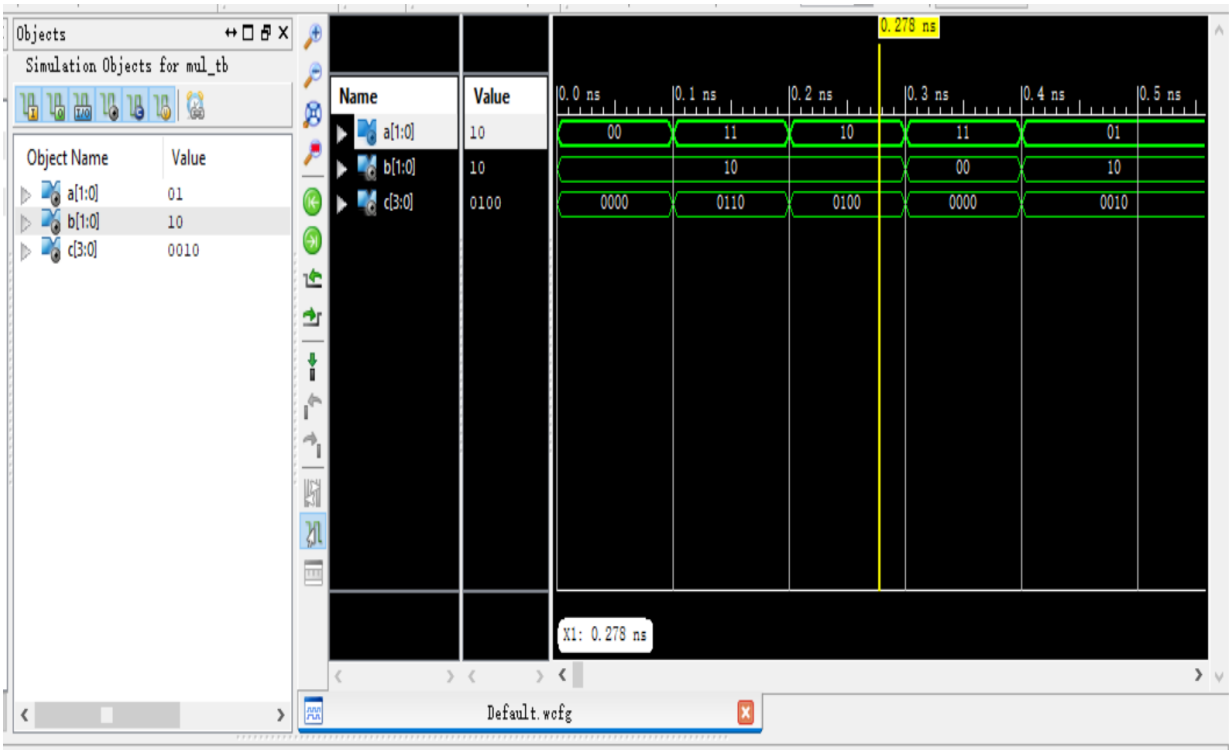


The logic circuit of a 2-bit multiplier

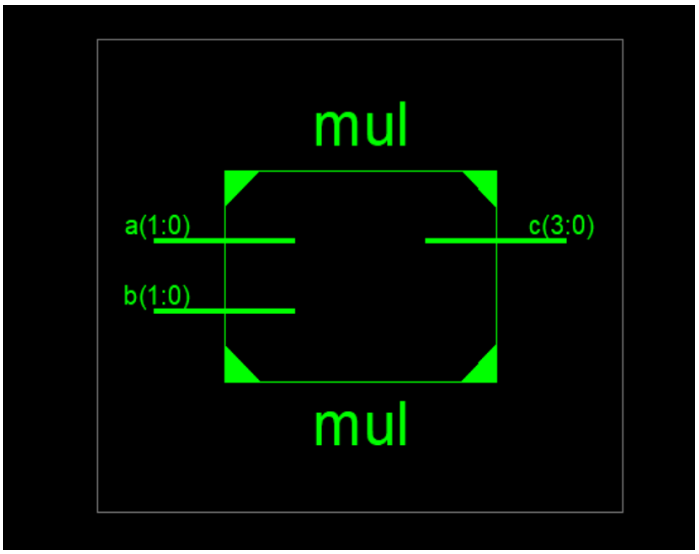
Code:

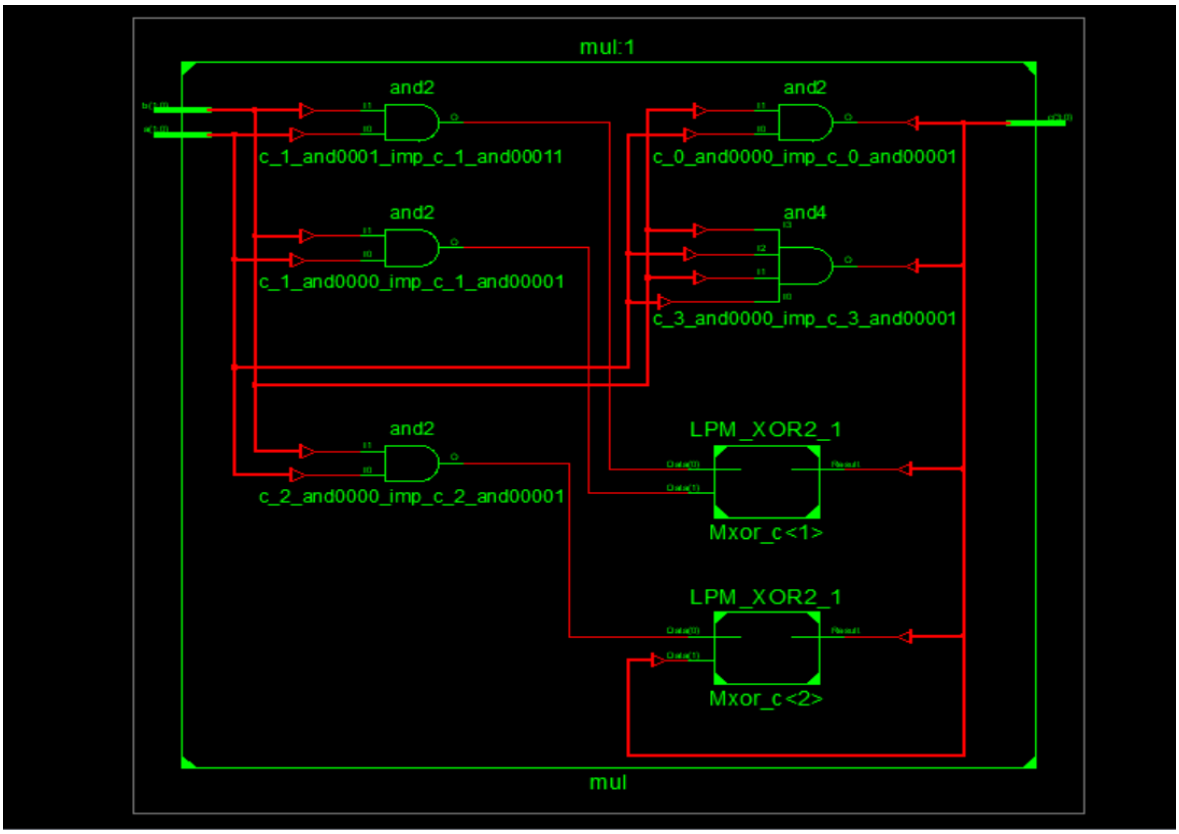
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity mul is
5      Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
6             b : in  STD_LOGIC_VECTOR (1 downto 0);
7             c : out  STD_LOGIC_VECTOR (3 downto 0)); --n^2 bits is needed to avoid overflow
8  end mul;
9
10 architecture Behavioral of mul is
11
12 begin
13
14 --according to the logic circuit of a 2 bit multiplier
15 c(0) <= a(0) and b(0);
16 c(1) <= (a(1) and b(0)) xor (a(0) and b(1));
17 c(2) <= ((a(1) and b(0)) and (a(0) and b(1))) xor (a(1) and b(1));
18 c(3) <= ((a(1) and b(0)) and (a(0) and b(1))) and (a(1) and b(1));
19
20 end Behavioral;
21
22
```

Testbench:

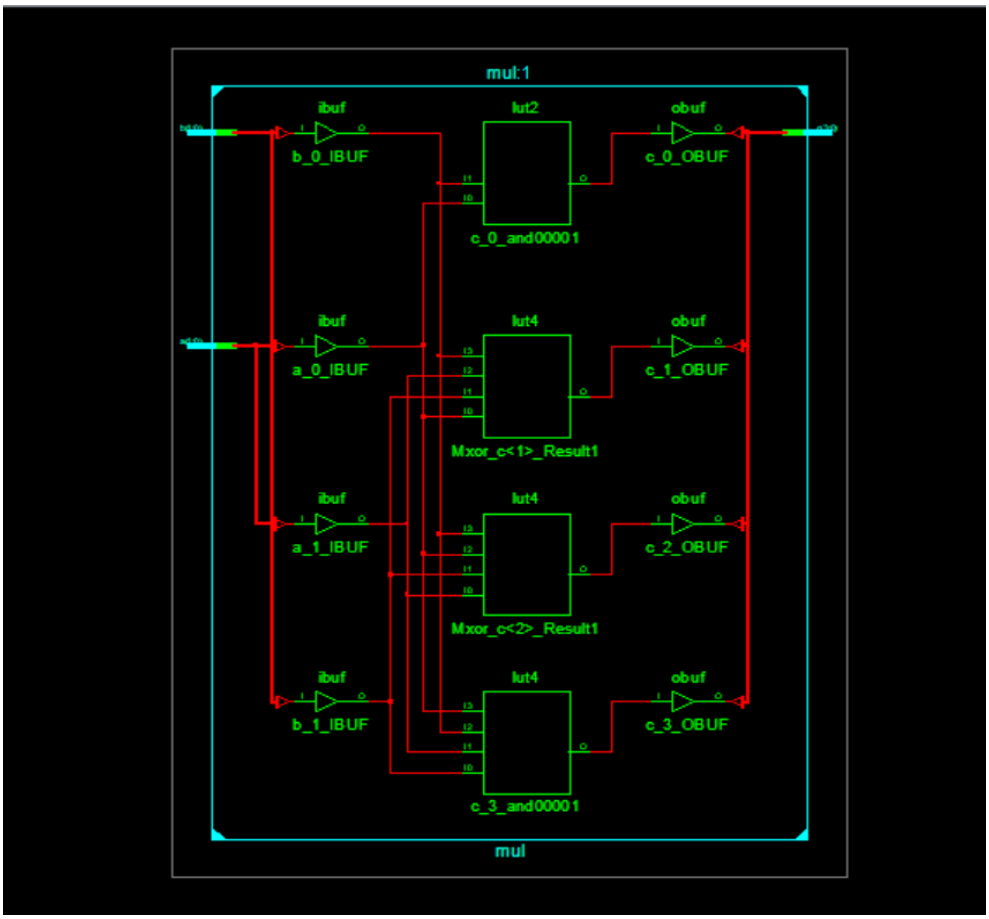


R TL Schematic:





Technology Schematic:



Register:

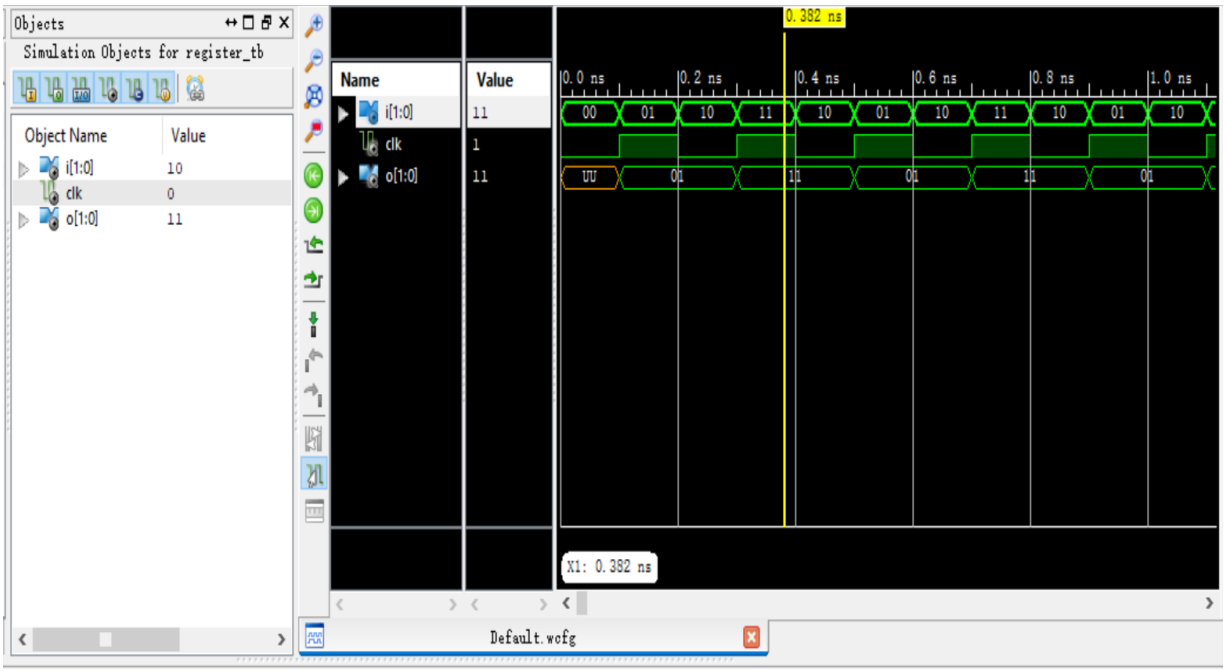
برای طراحی رجیستر دو بیتی (و 4 بیتی برای خروجی) از کد زیر استفاده کرده ایم که با یک فلیپ فلاپ D کار میکند. یعنی وقتی clk یک شد مقدار ورودی را از خود عبور میدهد. در طراحی میتوانستیم از پایه enable , reset هم استفاده کنیم که چون اشاره نشده بود استفاده نکردیم.

Code:

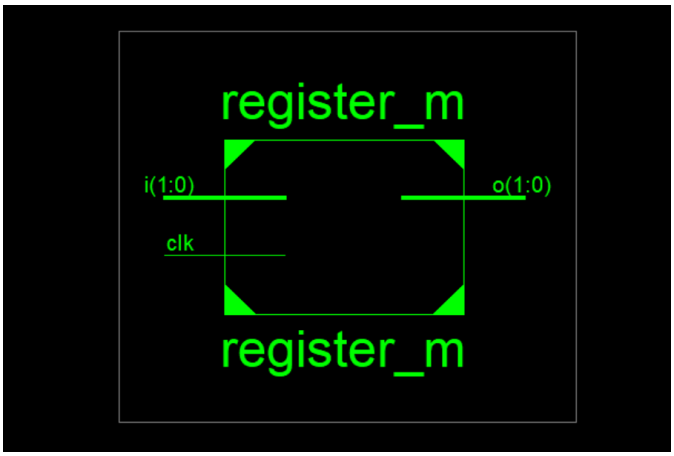
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity register_m is
5      Port ( i : in  STD_LOGIC_VECTOR (1 downto 0);
6             clk : in STD_LOGIC;
7             o : out  STD_LOGIC_VECTOR (1 downto 0));
8  end register_m;
9
10 architecture Behavioral of register_m is
11
12 begin
13  process(clk)
14  begin
15    if rising_edge (clk) then
16       o <= i;
17    end if;
18 end process;
19
20 end Behavioral;
21
22
```

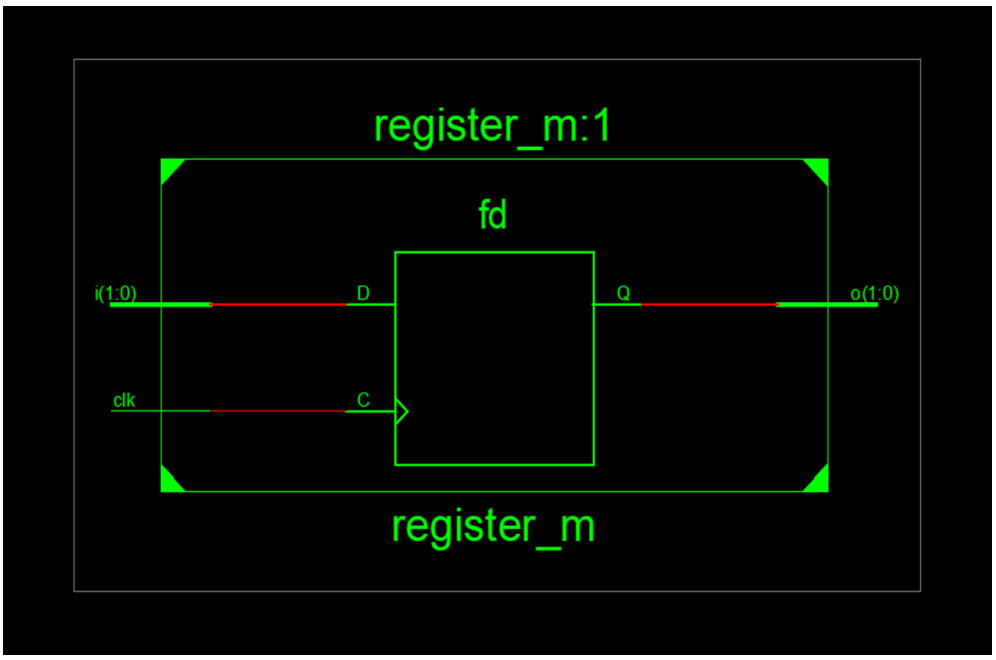
Testbench:

```
44    -- Stimulus process
45    stim_proc: process
46    begin
47    i<="00";
48    clk<='0';
49    wait for 100 ps;
50    i<="01";
51    clk<='1';
52    wait for 100 ps;
53    i<="10";
54    clk<='0';
55    wait for 100 ps;
56    i<="11";
57    clk<='1';
58    wait for 100 ps;
59    i<="10";
60    clk<='0';
61    wait for 100 ps;
62    i<="01";
63    clk<='1';
64    wait for 100 ps;
65    i<="10";
66    clk<='0';
67    wait for 100 ps;
68    i<="11";
69    clk<='1';
70    wait for 100 ps;
71    i<="10";
72    clk<='0';   wait for 100 ps;
73    i<="01";
74    clk<='1';
75    wait for 100 ps;
76    i<="10";
```

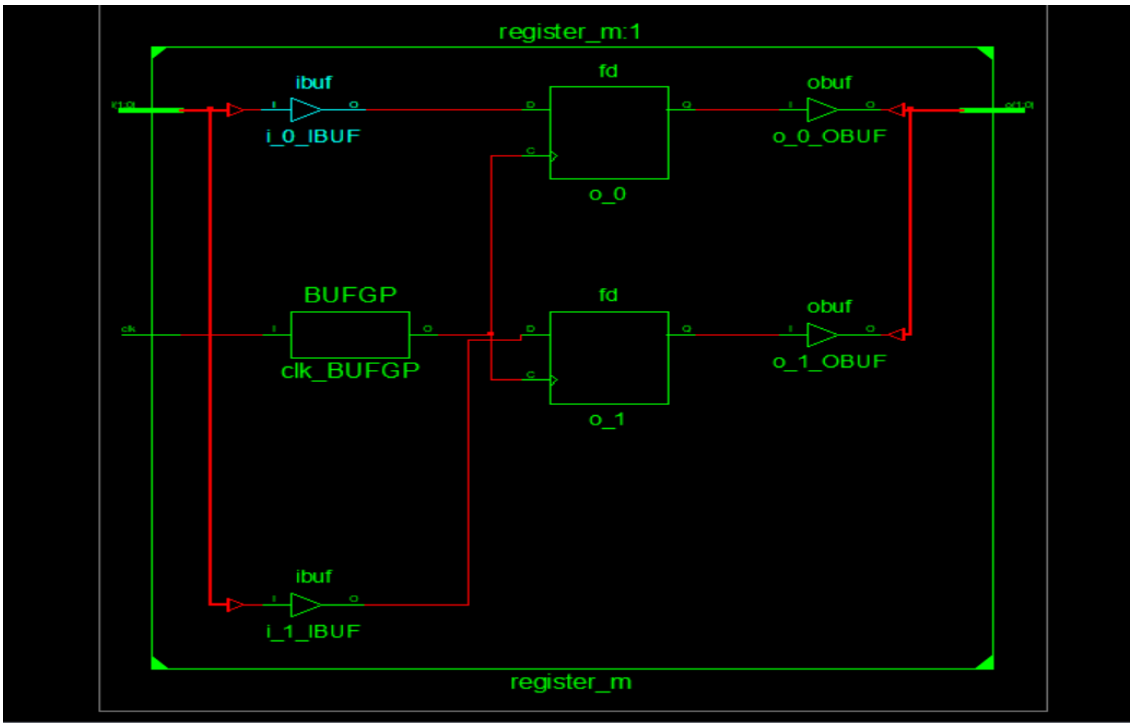


RTL Schematic:





Technology Schematic:



Main:

پس از قسمت architecture باید همه ی سیگنال ها و ماژول های تعریف شده در برنامه های مختلف (بصورت component) به کد اضافه شوند. سپس در قسمت اصلی برنامه برای نسبت دادن ورودی ها و خروجی های هر component به سیگنال تعریف شده portmap میکنیم (.U0,U1,...)

سپس در کد multiplexer با توجه به پایه select که یک ورودی دو بیتی است یکی از ماژول ها انتخاب میشوند. در این قسمت چون خروجی 4 بیتی در نظر گرفته شده است باید همه ی اعداد 4 بیتی شوند پس از نکته ی اشاره شده در قسمت Adding استفاده میکنیم. یعنی چون خروجی حاصل از کد sub وand دوبیتی هستند دو صفر و خروجی حاصل از adder سه بیتی است یک صفر به اولشان اضافه میکنیم.چون خروجی multiplier 4 بیتی است صفری به آن افزوده نمیشود.

مقدار flag های c و o را صفر میدهیم زیرا همانطور که در بخش sub بیان شد هرگز رخ نمیدهند.

سپس در قسمت مربوط به seven segment اعداد به صورت باینری نشان داده میشوند.

Code:

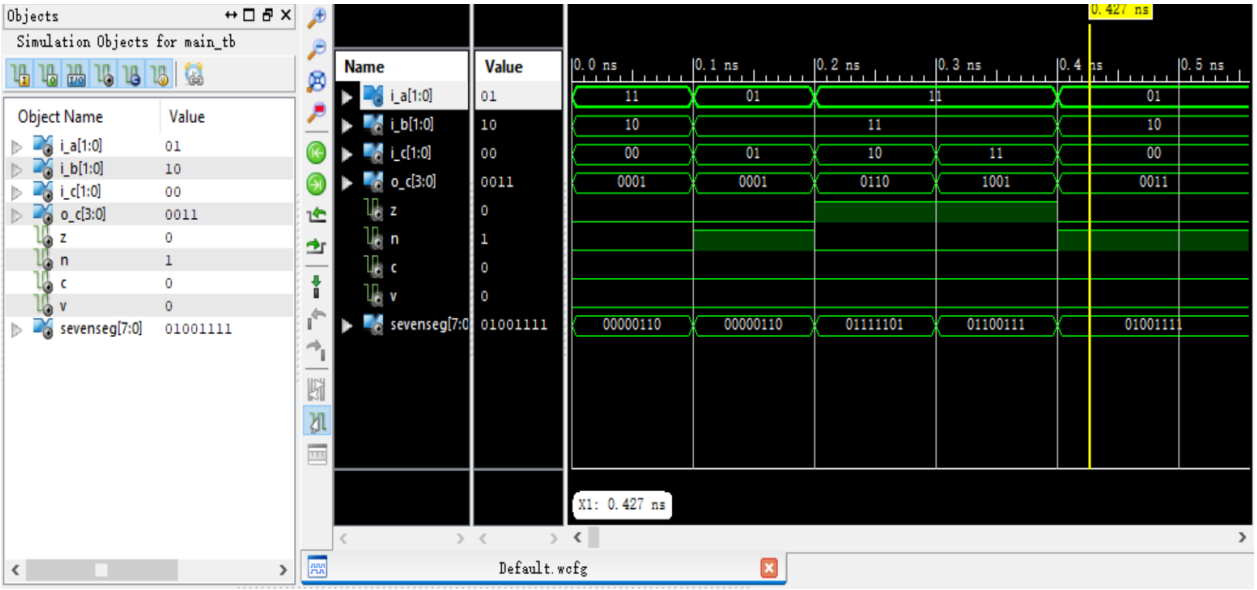
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity main_project is
    Port ( i_a : in  STD_LOGIC_VECTOR (1 downto 0);
           i_b : in  STD_LOGIC_VECTOR (1 downto 0);
           i_c : in  STD_LOGIC_VECTOR (1 downto 0);
           clk : in STD_LOGIC;
           o_c : out  STD_LOGIC_VECTOR (3 downto 0);
           z : out  STD_LOGIC; --zero flag
           n : out  STD_LOGIC; --negetive flag
           c : out  STD_LOGIC; --carry flag
           v : out  STD_LOGIC; --overflow flag
           sevenseg : out  STD_LOGIC_VECTOR (7 downto 0));
end main_project;

architecture Behavioral of main_project is

--describing the signals
signal subsignal : STD_LOGIC_VECTOR (1 downto 0);
signal andsignal : STD_LOGIC_VECTOR (1 downto 0);
signal addsignal : STD_LOGIC_VECTOR (2 downto 0);
signal mulsignal : STD_LOGIC_VECTOR (3 downto 0);
signal output : STD_LOGIC_VECTOR (3 downto 0);
signal a_i : STD_LOGIC_VECTOR (1 downto 0);
signal b_i : STD_LOGIC_VECTOR (1 downto 0);

--describing components
component sub
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           b : in  STD_LOGIC_VECTOR (1 downto 0);
           c : out  STD_LOGIC_VECTOR (1 downto 0);
           z : out  STD_LOGIC;
           n : out  STD_LOGIC);
end component;

component and_m
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           b : in  STD_LOGIC_VECTOR (1 downto 0);
           c : out  STD_LOGIC_VECTOR (1 downto 0));
end component;

component add
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           b : in  STD_LOGIC_VECTOR (1 downto 0);
           c : out  STD_LOGIC_VECTOR (2 downto 0));
end component;

component mul
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           b : in  STD_LOGIC_VECTOR (1 downto 0);
           c : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component register_m
    Port ( i : in  STD_LOGIC_VECTOR (1 downto 0);
           clk : in STD_LOGIC;
           o : out  STD_LOGIC_VECTOR (1 downto 0));
end component;

```

```
63  component register4
64      Port ( i : in  STD_LOGIC_VECTOR (3 downto 0);
65             clk : in  STD_LOGIC;
66             o : out  STD_LOGIC_VECTOR (3 downto 0));
67  end component;
68
69  begin
70  --port map
71  u0: sub port map
72  (
73     a => a_i,
74     b => b_i,
75     c => subsignal,
76     z => z,
77     n => n
78  );
79
80  u1: and_m port map
81  (
82     a => a_i,
83     b => b_i,
84     c => andsignal
85  );
86
87  u2: add port map
88  (
89     a => a_i,
90     b => b_i,
91     c => addsignal
92  );
93
94  u3: mul port map
95  (
96     a => a_i,
97     b => b_i,
98     c => mulsignal
99  );
100
101 u4: register_m port map
102 (
103    i => i_a,
104    clk => clk,
105    o => a_i
106 );
107
108 u5: register_m port map
109 (
110    i => i_b,
111    clk => clk,
112    o => b_i
113 );
114
115 u6: register4 port map
116 (
117    i => output,
118    clk => clk,
119    o => o_c
120 );
121
122 --multiplexer
123  process(i_c,subsignal,andsignal,addsignal, mulsignal)
124  begin
125    case i_c is
126       when "00" =>
127             output <= ("00" & subsignal);
128       when "01" =>
129             output <= ("00" & andsignal);
130       when "10" =>
131             output <= ('0' & addsignal);
132       when "11" =>
133             output <= mulsignal;
134       when others => null;
135    end case;
136 end process;
137
138 v <= '0';
139 c <= '0';
140 --pp
141 --seven segment
142 process(output)
143 begin
144     case output is
145       when x"0" =>
146                      sevenseg <= x"3F";
147       when x"1" =>
148                      sevenseg <= x"06";
149       when x"2" =>
150                      sevenseg <= x"5B";
151       when x"3" =>
152                      sevenseg <= x"4F";
```

```
153         when x"4" =>
154                         sevenseg <= x"66";
155         when x"5" =>
156                         sevenseg <= x"6D";
157         when x"6" =>
158                         sevenseg <= x"7D";
159         when x"7" =>
160                         sevenseg <= x"07";
161         when x"8" =>
162                         sevenseg <= x"7F";
163         when x"9" =>
164                         sevenseg <= x"67";
165         when x"A" =>
166                         sevenseg <= x"77";
167         when x"B" =>
168                         sevenseg <= x"7C";
169         when x"C" =>
170                         sevenseg <= x"39";
171         when x"D" =>
172                         sevenseg <= x"5E";
173         when x"E" =>
174                         sevenseg <= x"79";
175         when x"F" =>
176                         sevenseg <= x"71";
177         when others =>
178                         sevenseg <= x"00";
179     end case;
180 end process;
181
182 end Behavioral;
183
184
```

Testbench:

```
57
58    -- Stimulus process
59    stim_proc: process
60    begin
61
62    i_a<="11";
63    i_b<="10";
64    i_c<="00";
65    wait for 100 ps;
66    i_a<="01";
67    i_b<="11";
68    i_c<="01";
69    wait for 100 ps;
70    i_a<="11";
71    i_b<="11";
72    i_c<="10";
73    wait for 100 ps;
74    i_a<="11";
75    i_b<="11";
76    i_c<="11";
77    wait for 100 ps;
78    i_a<="01";
79    i_b<="10";
80    i_c<="00";
81    wait for 100 ps;
82
83
84       wait;
85    end process;
86
87 END;
88
```



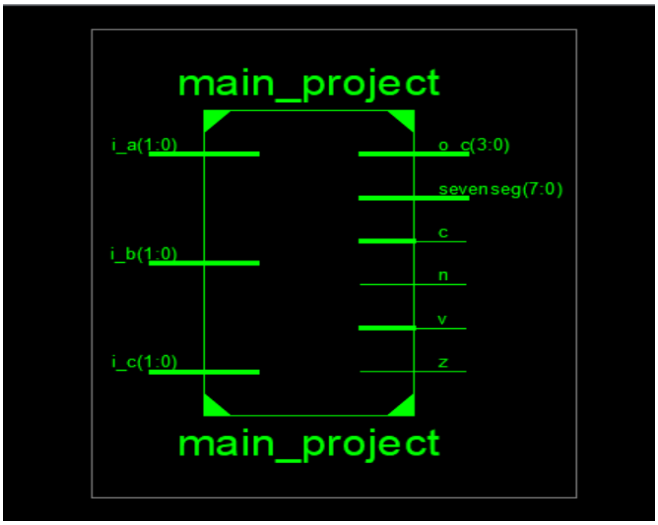
همینطور که در تصویر بالا مشاهده میشود به ازای i_c = 00 انتظار داریم عمل تفریق انجام شود یعنی i_a - i_b =11-10=0001 که به درستی صورت گرفته است.

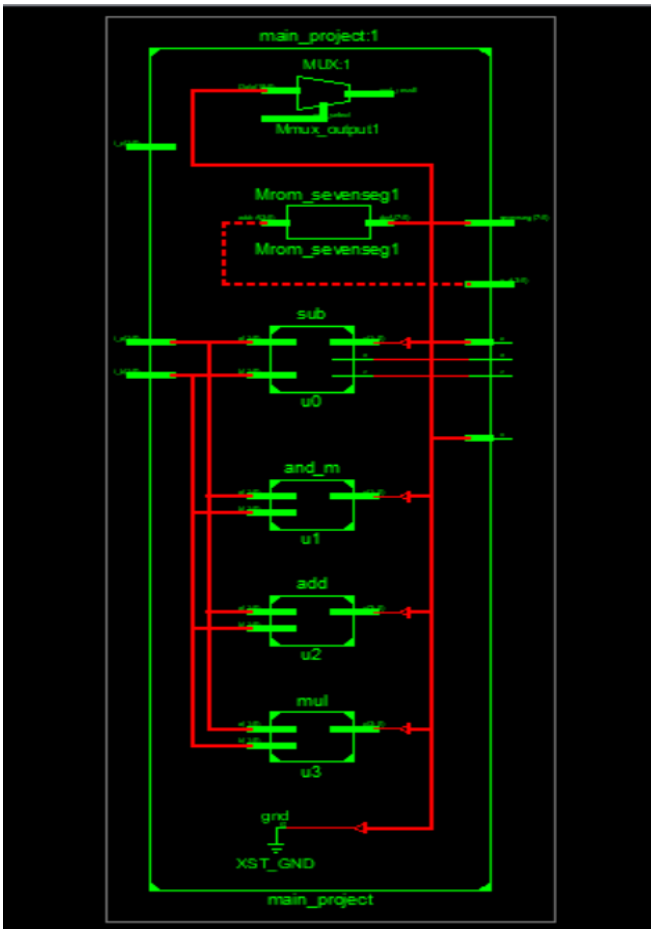
به ازای i_c = 01 انتظار داریم عمل and انجام شود یعنی i_a and i_b = 01 and 11 = 0001 که به درستی صورت گرفته است.

به ازای i_c=10 انتظار داریم عمل جمع انجام شود یعنی i_a + i_b =11+11=0110 که به درستی صورت گرفته است.

به ازای i_c=11 انتظار داریم عمل ضرب انجام شود یعنی i_a * i_b = 11*11=1001 که به درستی صورت گرفته است.

RTL Schematic:





Technology Schematic: