محمد مهدی نظری ۹۹۳۱۰۶۱ – آرمین ابراهیمی صبا ۹۹۳۱۰۸۶

برای کاهش حجم گزارش کار و همچنین به دلیل اینکه در آزمایش قبلی بررسی شد از آوردن کد و تست گیت های or ، and و xor خودداری شده همچنین با توجه به آزمایش مربوطه گیت ۴ ورودی or به نام or_4 و گیت ۳ ورودی and به نام and_3 از آن ها entity ساخته شده و در آزمایش مربوطه کامپوننت آن ها تعریف شده و از آنها مثال یا شی ساخته شده و سپس طبق شکل مدار مربوطه مورد استفاده قرار گرفته اند .

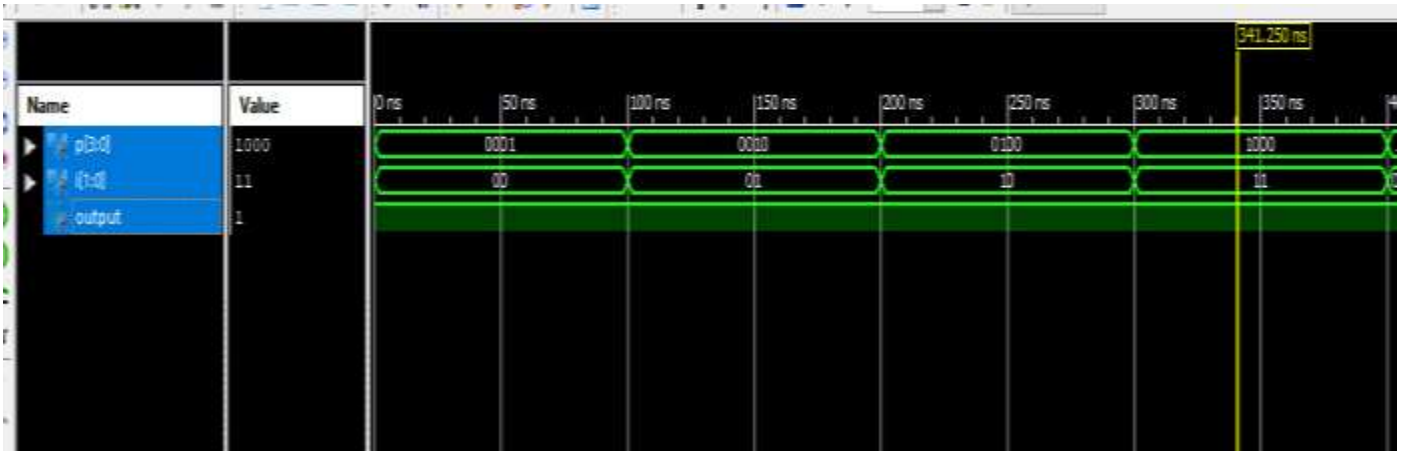
در تصویر کد مالتی پلکسر و دیکدر و انکودر ورودی و خروجی برای فهم بیشتر تک بیت جدا داده شده اند اما در تست بنچ مربوطه برای ساده و قابل فهم تر شدن جواب تست ورودی و خروجی به شکل وکتور در نظر گرفته شده .

۱ – مالتی پلکسر

میدانیم مالتی پلکسر ۴ به ۱ ، ۴ بیت ورودی (در این کد p) و دوبیت کلید دارد (در این کد i) که بیت متناظر کلید در ورودی به خروجی انتقال میکند مثلا اگر کلید 01 باشد که معادل ۱ دسیمال است پس بیت ۱ ورودی (یعنی p(1)) به خروجی انتقال پیدا میکند . در تست بنچ هم هر چهار حالت کلید یعنی از ۰ تا ۳ چک شده و بیت متناظر در ورودی ۱ و بقیه ۰ در نظر گرفته شده تا خروجی همیشه ۱ شود که بدین معنی است که آن بیت مربوطه فقط به خروجی منتقل میشود.

```
32  entity MUX is
33      Port ( p0 : in STD_LOGIC;
34             p1 : in STD_LOGIC;
35             p2 : in STD_LOGIC;
36             p3 : in STD_LOGIC;
37             i0 : in STD_LOGIC;
38             i1 : in STD_LOGIC;
39             output : out STD_LOGIC);
40  end MUX;
41
42  architecture Behavioral of MUX is
43      component and_3 is
44          port(
45              a: in std_logic;
46              b: in std_logic;
47              c: in std_logic;
48              output: out std_logic);
49      end component and_3;
50      component or_4 is
51          port(
52              a: in std_logic;
53              b: in std_logic;
54              c: in std_logic;
55              d: in std_logic;
56              output: out std_logic);
57      end component or_4;
58      signal not_i0, not_i1: std_logic;
59      signal y1, y2, y3, y0: std_logic;
60  begin
61  not_i0 <= not i0;
62  not_i1 <= not i1;
63  and_3_inst01: and_3 port map(a => i1, b => i0, c => p3, output => y3);
64  and_3_inst02: and_3 port map(a => i1, b => not_i0, c => p2, output => y2);
65  and_3_inst03: and_3 port map(a => not_i1, b => i0, c => p1, output => y1);
66  and_3_inst04: and_3 port map(a => not_i1, b => not_i0, c => p0, output => y0);
67  or_inst: or_4 port map(a => y0, b => y1, c => y2, d => y3, output => output);
68  end Behavioral;
```

```
31
32  entity MUX_tb is
33  end MUX_tb;
34
35  architecture test of MUX_tb is
36  component MUX is
37  Port ( p : in STD_LOGIC_vector(3 downto 0 );
38             i : in STD_LOGIC_vector(1 downto 0 );
39             output : out STD_LOGIC);
40  end component ;
41  signal p :  STD_LOGIC_vector(3 downto 0 );
42  signal i :  STD_LOGIC_vector(1 downto 0 );
43  signal  output : STD_LOGIC ;
44  begin
45  MUX_instanece : MUX port map (p => p , i=>i , output =>output );
46  stim_proc : process
47  begin
48  p <= "0001";
49  i<= "00" ;
50  wait for 100 ns ;
51
52  p <= "0010";
53  i<= "01" ;
54  wait for 100 ns ;
55
56  p <= "0100";
57  i<= "10" ;
58  wait for 100 ns ;
59
60  p <= "1000";
61  i<= "11" ;
62  wait for 100 ns ;
63  end process ;
64
65  end test;
66
67
```

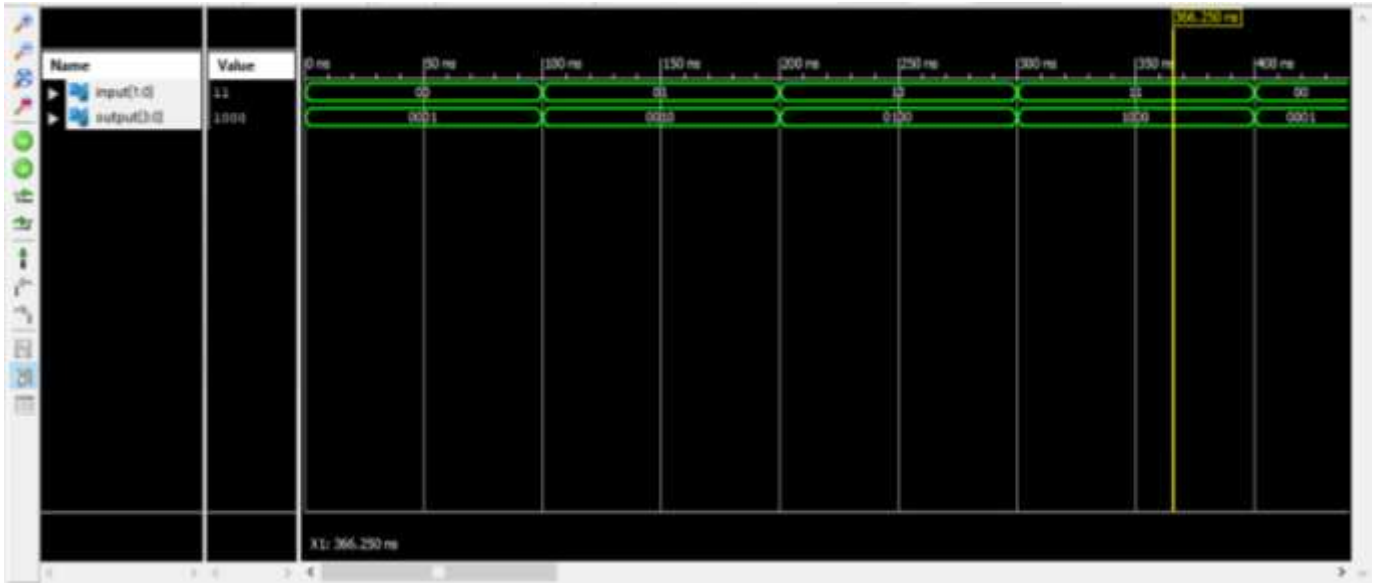


۲ - دیکدر

در دیکدر ۲ به ۴ ، ۲بیت ورودی داریم که بیت متناظر با عدد ورودی در خروجی ۴ بیتی ۱ و بقیه صفر میشوند مثلا به ازای ورودی ۰۱ خروجی ۰۰۱۰ میگیریم . در تست بنچ هر چهار حالت ورودی چک شده .

```
entity Decoder is
    Port ( in0 : in STD_LOGIC;
           in1 : in STD_LOGIC;
           out0 : out STD_LOGIC;
           out1 : out STD_LOGIC;
           out2 : out STD_LOGIC;
           out3 : out STD_LOGIC);
end Decoder;

architecture Behavioral of Decoder is
    component and_gate is
        port(
            a: in std_logic;
            b: in std_logic;
            output: out std_logic);
    end component and_gate;
    signal not_in0, not_in1: std_logic;
begin
not_in0 <= not in0;
not_in1 <= not in1;
and_inst01: and_gate port map(a => in1, b => in0, output => out3);
and_inst02: and_gate port map(a => in1, b => not_in0, output => out2);
and_inst03: and_gate port map(a => not_in1, b => in0, output => out1);
and_inst04: and_gate port map(a => not_in1, b => not_in0, output => out0);

end Behavioral;
```

```
entity Decoder_tb is
end Decoder_tb;

architecture test of Decoder_tb is
component Decoder is
 Port ( input : in STD_LOGIC_vector(1 downto 0);
           output : out STD_LOGIC_vector(3 downto 0)
           );
end component ;
signal input :  STD_LOGIC_vector(1 downto 0);
signal output :  STD_LOGIC_vector(3 downto 0);
begin
Decoder_instance : Decoder port map ( input => input , output => output );
stim_proc : process
begin
input <= "00" ;
wait for 100 ns ;

input <= "01" ;
wait for 100 ns ;

input <= "10" ;
wait for 100 ns ;

input <= "11" ;
wait for 100 ns ;
end process ;


end test;
```

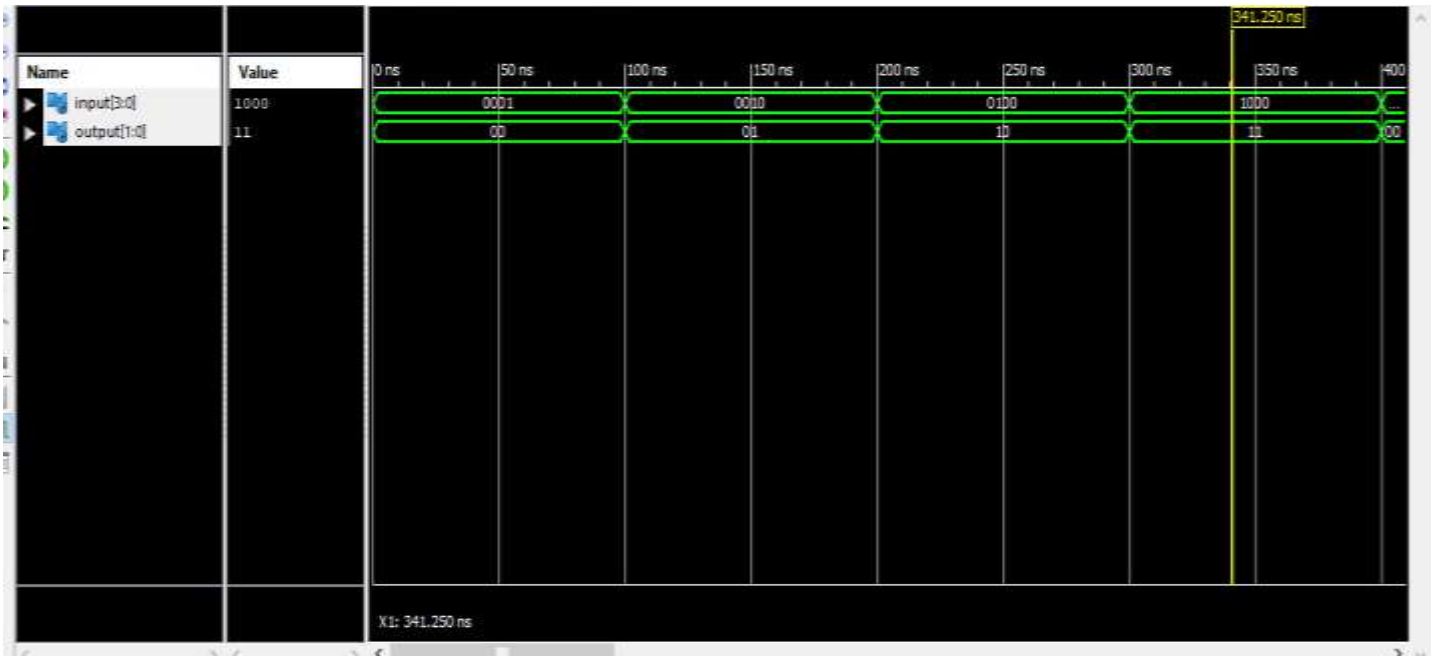


۳ – انکودر

انکودر دقیقا برعکس دیکودر است که شماره بیت ۱ در ورودی ۴ بیتی به عدد دوبیتی باینری خروجی تبدیل میشود مثلا ورودی ۱۰۰۰ که شماره بیت ۱ عدد ۳ دسیمال است در خروجی معادل باینری آن یعنی ۱۱ تولید میشود . همانند دیکودر چهارحالت ورودی در تست بنچ بررسی شده .

```
entity Encoder is
    Port ( in0 : in STD_LOGIC;
           in1 : in STD_LOGIC;
           in2 : in STD_LOGIC;
           in3 : in STD_LOGIC;
           out0 : out STD_LOGIC;
           out1 : out STD_LOGIC);
end Encoder;

architecture Behavioral of Encoder is
    component or_gate is
        port(
            a: in std_logic;
            b: in std_logic;
            output: out std_logic);
    end component or_gate;
begin
or_inst01: or_gate port map(a => in3, b => in2, output => out1);
or_inst02: or_gate port map(a => in3, b => in1, output => out0);
end Behavioral;
```

```
entity Encoder_tb is
end Encoder_tb;

architecture test of Encoder_tb is
component Encoder is
 Port ( input : in STD_LOGIC_vector(3 downto 0 );
          output : out STD_LOGIC_vector( 1 downto 0 )
          );
          end component ;
         signal  input :  STD_LOGIC_vector(3 downto 0 );
          signal output :  STD_LOGIC_vector( 1 downto 0 );
begin
Encoder_instance : Encoder port map ( input => input , output => output );
stim_proc : process
begin
input <= "0001" ;
wait for 100 ns ;

input <= "0010" ;
wait for 100 ns ;

input <= "0100" ;
wait for 100 ns ;

input <= "1000" ;
wait for 100 ns ;
end process ;

end test;
```



۴ – مقایسه کننده ۴ بیت

ابتدا مقایسه گر تک بیت مانند شکل پیش گزارش طراحی شده و از به هم وصل کردن ۴ مقایسه کننده تک بیت همانند طرح داخل پیش گزارش مقایسه کننده ۴ بیت طراحی شده . در مقایسه کننده های تک بیت سه ورودی دیگر علاوه بر دوبیت که باید مقایسه شوند وجوددارند (lt , eq , gt)(به ترتیب از راست به این معنی هستند که بیت های قبلی بزرگتر ، مساوی و کوچکتر هستند) که وضعیت مقایسه بیت های کوچکتر را مشخص میکنند که در تعیین خروجی نهایی دوبیت بزرگتر تاثیر گذارند (شکل مدار آن در پیش گزارش موجود است) .

```
entity one_bit_comp is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           eq : in STD_LOGIC;
           gt : in STD_LOGIC;
           lt : in STD_LOGIC;
           a_gt_b : out STD_LOGIC;
           a_lt_b : out STD_LOGIC;
           a_eq_b : out STD_LOGIC);
end one_bit_comp;

architecture Behavioral of one_bit_comp is
    component and_gate is
        port(
            a: in std_logic;
            b: in std_logic;
            output: out std_logic);
    end component and_gate;
    component or_gate is
        port(
            a: in std_logic;
            b: in std_logic;
            output: out std_logic);
    end component or_gate;
    component xor_gate is
        port(
            a: in std_logic;
            b: in std_logic;
            output: out std_logic);
    end component xor_gate;
    signal a_and_not_b : std_logic;
    signal b_and_not_a : std_logic;
    signal not_b : std_logic;
    signal not_a : std_logic;
    signal a_xor_b : std_logic;
    signal a_xnor_b : std_logic;
    signal gt_and_eq : std_logic;
    signal lt_and_eq : std_logic;
begin
    not_b <= not b;
    not_a <= not a;
    and_inst01: and_gate port map(a => a, b => not_b, output => a_and_not_b);
    and_inst02: and_gate port map(a => not_a, b => b, output => b_and_not_a);
    xor_inst: xor_gate port map(a => a, b => b, output => a_xor_b);
    a_xnor_b <= not a_xor_b;
    and_inst03: and_gate port map(a => gt, b => a_xnor_b, output => gt_and_eq);
    and_inst04: and_gate port map(a => lt, b => a_xnor_b, output => lt_and_eq);
    and_inst05: and_gate port map(a => a_xnor_b, b => eq, output => a_eq_b);
    or_inst01: or_gate port map(a => a_and_not_b, b => gt_and_eq, output => a_gt_b);
    or_inst02: or_gate port map(a => b_and_not_a, b => lt_and_eq, output => a_lt_b);
end Behavioral;
```

```
entity four_bit_comp is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (3 downto 0);
           a_lt_b : out STD_LOGIC;
           a_gt_b : out STD_LOGIC;
           a_eq_b : out STD_LOGIC);
end four_bit_comp;

architecture Behavioral of four_bit_comp is
    component one_bit_comp is
        Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           eq : in STD_LOGIC;
           gt : in STD_LOGIC;
           lt : in STD_LOGIC;
           a_gt_b : out STD_LOGIC;
           a_lt_b : out STD_LOGIC;
           a_eq_b : out STD_LOGIC);
    End component one_bit_comp;
    signal init_sigs: std_logic_vector(2 downto 0) := "010";
    signal sigs: std_logic_vector (8 downto 0);
begin
one_bit_comp_instance0: one_bit_comp port map(a => a(0), b => b(0), eq => init_sigs(1),
 gt => init_sigs(0), lt => init_sigs(2), a_gt_b => sigs(0), a_lt_b => sigs(1), a_eq_b => sigs(2));

one_bit_comp_instance1: one_bit_comp port map(a => a(1), b => b(1), eq => sigs(2),
 gt => sigs(0), lt => sigs(1), a_gt_b => sigs(3), a_lt_b => sigs(4), a_eq_b => sigs(5));

one_bit_comp_instance2: one_bit_comp port map(a => a(2), b => b(2), eq => sigs(5),
 gt => sigs(3), lt => sigs(4), a_gt_b => sigs(6), a_lt_b => sigs(7), a_eq_b => sigs(8));

one_bit_comp_instance3: one_bit_comp port map(a => a(3), b => b(3), eq => sigs(8),
 gt => sigs(6), lt => sigs(7), a_gt_b => a_gt_b, a_lt_b => a_lt_b, a_eq_b => a_eq_b);

end Behavioral;
```

تست بنچ مقایسه گر ۴ بیت :

در این قسمت هر ۳ حالت ممکن برگتر – مساوی – کوچکتر برای ورودی a نسبت به b چک شده است .

```
35  architecture test of four_bit_comp_tb is
36  component four_bit_comp is
37  Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
38             b : in STD_LOGIC_VECTOR (3 downto 0);
39             a_lt_b : out STD_LOGIC;
40             a_gt_b : out STD_LOGIC;
41             a_eq_b : out STD_LOGIC);
42  end component ;
43  signal a :  STD_LOGIC_VECTOR (3 downto 0);
44  signal b :  STD_LOGIC_VECTOR (3 downto 0);
45   signal a_lt_b :  STD_LOGIC;
46   signal a_gt_b :  STD_LOGIC;
47   signal a_eq_b :  STD_LOGIC;
48  begin
49  four_bit_comp_instance : four_bit_comp port map (a=>a ,b=>b  ,a_lt_b=>a_lt_b ,a_gt_b=>a_gt_b,a_eq_b=>a_eq_b);
50  stim_proc :process
51  begin
52  a <= "1100";
53  b<= "0011";
54  wait for 100 ns ;
55
56  a <= "1111";
57  b<= "1111";
58  wait for 100 ns ;
59
60  a <= "1100";
61  b<= "1111";
62  wait for 100 ns ;
63
64  end process ;
65
66  end test;
67
68
```