"به نام یزدان پاک"

گزارش کار آزمایش چهارم

اعضای گروه:

کیانا آقا کثیری 9831006

محمد چوپان 9831125

سارا تاجرنیا 9831016

نویسنده گزارش : محمد چوپان

تاریخ آزمایش : 1400/2/1

تاریخ تحویل گزارش :1400/2/7

شرح آزمایش:

در این آزمایش، یک جمع‌کننده ۴ بیتی از نوع‌های زیر، طراحی و پیاده‌سازی گردد. طراحی انجام شده با زبان توصیف سخت‌افزاری VHDL و در **سطح تجرید گیت** انجام شود. در این پیاده‌سازی از واحدهای Full Adder و HalfAdder استفاده گردد. تهیه TestBench برای این جمع کننده نیز جزء الزام‌های آزمایش است.

الف) Ripple Adder (Cascaded Adder)

ب) Carry-Lookahead Adder

ج) Carry Select Adder

شکل ۱ : **شرح آزمایش**

هدف از این آزمایش، آشنایی با چگونگی عملکرد هر یک از جمع‌کننده‌ها است. تفاوت این جمع‌کننده‌ها در سرعت محاسبه عملیات جمع می‌باشد. در این آزمایش هدف آشنایی با نحوه عملکرد و پیاده‌سازی سخت‌افزاری آن‌ها در **سطح تجرید گیت** است.

شکل ۲ : **هدف آزمایش**

خروجی‌های مورد انتظار آزمایش:

هر یک از موارد زیر باید تحویل داده شود:

- طراحی شماتیک طرح (محتوای توصیف) بر روی کاغذ
- بررسی درستی سیگنال‌های خروجی مدار مورد نظر با انجام شبیه‌سازی
- پیاده‌سازی بر روی FPGA و نیز اطمینان از درستی عملکرد آن با استفاده از ورودی‌ها و خروجی‌های قابل استفاده روی برد FPGA
- تحلیل و مقایسه سرعت عملکرد جمع کننده‌های فوق با یکدیگر و در عرض بیت‌های مختلف

شکل ۳ : **خروجی مورد انتظار آزمایش**

برای ساخت یک جمع کننده ها به full adder یک بیتی نیاز داریم که خود full adder ها نیز با استفاده از half adder پیاده سازی شده اند.

لذا ابتدا باید full adder و half adder ها را پیاده سازی کنیم.

Half Adder:

sum = A xor B delay(sum) = d

carry = A and B delay(carry) = d

cost = 2g

behavioral of half adder:

```
1  library IEEE;
2
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  use IEEE.STD_LOGIC_ARITH.ALL;
6
7  use IEEE.STD_LOGIC_UNSIGNED.ALL;
8
9
10 entity HA is
11
12 Port ( A,B : in  STD_LOGIC;
13
14        S,C : out  STD_LOGIC);
15
16 end HA;
17
18
19 architecture dataflow of HA is
20
21
22 begin
23
24 S <= A XOR B;
25
26 C <= A AND B;
27
28 end dataflow;
29
```

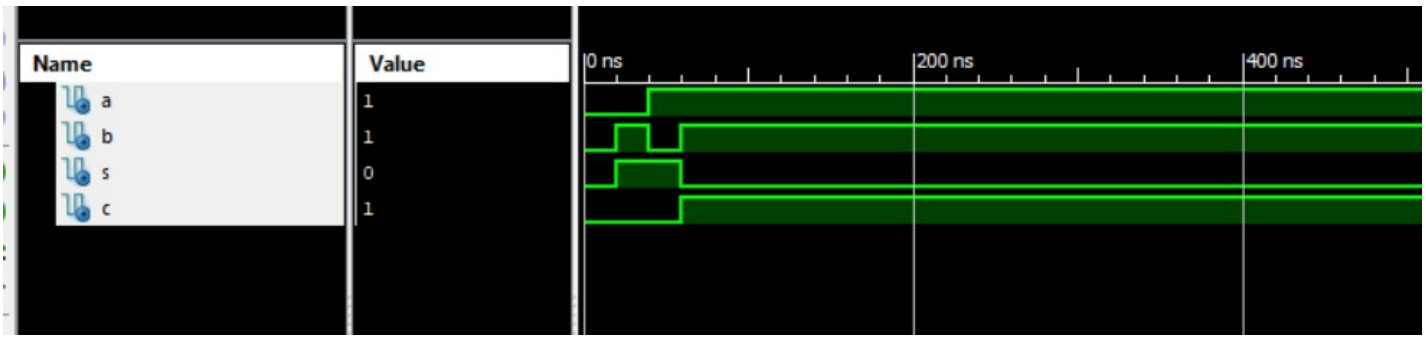
شکل 4 : توصیف **half adder**

Test Bench:

```
uut: HA port map(
a => a, b => b,
s => s,
c => c);

stim: process
begin

a <= '0';
b <= '0';
wait for 20 ns;

a <= '0';
b <= '1';
wait for 20 ns;

a <= '1';
b <= '0';
wait for 20 ns;

a <= '1';
b <= '1';
wait for 20 ns;

wait;

end process;

end tb;
```

شکل ۵ : تست بنچ **half adder**

Result of simulation in isim:



شکل 6 : نتایج شبیه سازی **half adder**

full Adder:

sum = A xor B xor C OR S0 = HA(A, B) S=HA(S0, Cin) delay(sum) = d

carry = (A and B) or (C and B) or (A and C) OR carry = S1 or S2 delay(carry) = 2d

cost = 5g

behavioral of half adder:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity FA is
7  Port ( A, B, Cin : in STD_LOGIC;
8         S, Cout : out STD_LOGIC);
9  end FA;
10
11 architecture structural of FA is
12
13 component HA is
14 Port ( A,B : in STD_LOGIC;
15        S,C : out STD_LOGIC);
16 end component;
17
18
19 SIGNAL S0,S1,S2:STD_LOGIC;
20
21 begin
22
23 U1:HA PORT MAP(A=> A,B=> B,S=>S0,C=>S1);
24 U2:HA PORT MAP(A=> S0,B=> Cin,S=> S,C=>S2);
25 Cout <= S2 or S1;
26
27 end structural;
28
```

شکل 7 : توصیف **full adder**

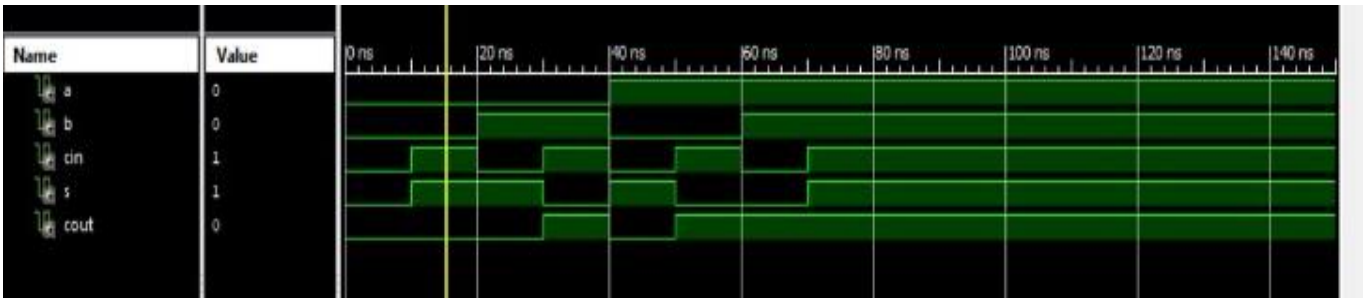
Test Bench:

```
 7  architecture tb of fa_tb is
 8  component FA is
 9  Port (   A, B, Cin : in STD_LOGIC;
10     S, Cout : out STD_LOGIC);
11  end component;
12
13  signal A, B, Cin, S, Cout : STD_LOGIC;
14
15  begin
16
17  uut : FA port map(
18  A => A, B => B, Cin => Cin, S => S, Cout => Cout);
19
20  stim : process
21  begin
22
23  A <= '0';
24  B <= '0';
25  Cin <= '0';
26  wait for 10 ns;
27  assert ((S = '0') and (Cout = '0'))
28  report "test failed for input combination 000" severity error;
29
30  A <= '0';
31  B <= '0';
32  Cin <= '1';
33  wait for 10 ns;
34  assert ((S = '1') and (Cout = '0'))
35  report "test failed for input combination 001" severity error;
36
37  A <= '0';
38  B <= '1';
39  Cin <= '0';
40  wait for 10 ns;
41  assert ((S = '1') and (Cout = '0'))
42  report "test failed for input combination 010" severity error;
43
```

```
44  A <= '0';
45  B <= '1';
46  Cin <= '1';
47  wait for 10 ns;
48  assert ((S = '0') and (Cout = '1'))
49  report "test failed for input combination 011" severity error;
50
51  A <= '1';
52  B <= '0';
53  Cin <= '0';
54  wait for 10 ns;
55  assert ((S = '1') and (Cout = '0'))
56  report "test failed for input combination 100" severity error;
57
58  A <= '1';
59  B <= '0';
60  Cin <= '1';
61  wait for 10 ns;
62  assert ((S = '0') and (Cout = '1'))
63  report "test failed for input combination 101" severity error;
64
65  A <= '1';
66  B <= '1';
67  Cin <= '0';
68  wait for 10 ns;
69  assert ((S = '0') and (Cout = '1'))
70  report "test failed for input combination 110" severity error;
71
72  A <= '1';
73  B <= '1';
74  Cin <= '1';
75  wait for 10 ns;
76  assert ((S = '1') and (Cout = '1'))
77  report "test failed for input combination 111" severity error;
78  wait;
```

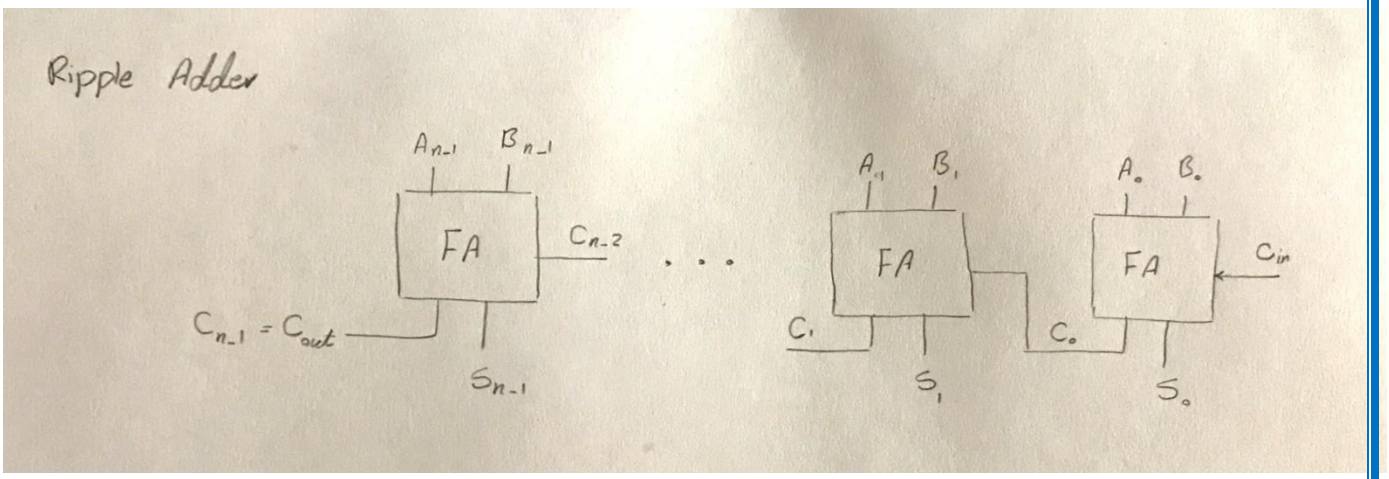
شکل 8 : تست بنچ **full adder**

Result of simulation in isim:



شکل 9: نتایج شبیه سازی **full adder**

الف) Ripple adder:



شکل 10 : **طراحی شماتیک ripple adder**

delay(sum) = (2n – 1)d

delay(carry) = 2nd

cost = 5ng

ripple adder به این صورت کار میکند که ابتدا بیت های A0 , B0 و Cin را با استفاده از FA با هم جمع میکند سپس خروجی S0 را تولید کرده و Cout تولید شده را به عنوان Cin به FA بعدی میدهد و به همین صورت پیش میرود تا تمام بیت های خروجی S و Cout تهایی تولید شود.

behavioral of Ripple adder:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Ripple_Adder is
5  Port (   A : in STD_LOGIC_VECTOR (3 downto 0);
6     B : in STD_LOGIC_VECTOR (3 downto 0);
7     Cin : in STD_LOGIC;
8     S : out STD_LOGIC_VECTOR (3 downto 0);
9     Cout : out STD_LOGIC);
10 end Ripple_Adder;
11
12 architecture Behavioral of Ripple_Adder is
13
14
15 component FA
16 Port (  A : in STD_LOGIC;
17    B : in STD_LOGIC;
18    Cin : in STD_LOGIC;
19    S : out STD_LOGIC;
20    Cout : out STD_LOGIC);
21 end component;
22
23 signal c1,c2,c3: STD_LOGIC;
24
25 begin
26
27  -- Port Mapping Full Adder 4 times
28 FA1: FA port map( A(0), B(0), Cin, S(0), c1);
29 FA2: FA port map( A(1), B(1), c1, S(1), c2);
30 FA3: FA port map( A(2), B(2), c2, S(2), c3);
31 FA4: FA port map( A(3), B(3), c3, S(3), Cout);
32
33 end Behavioral;
```

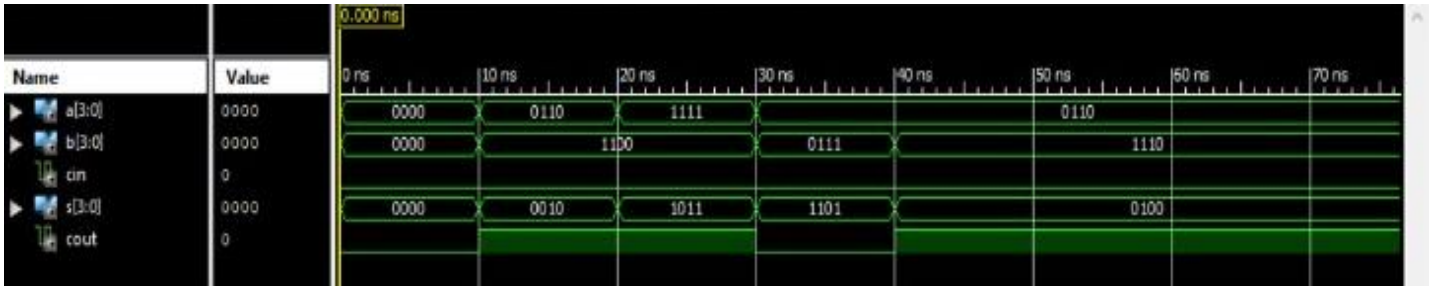
شکل 11 : توصیف **ripple adder 4 bit با استفاده از ۴ full adder**

Test Bench:

```
10  COMPONENT Ripple_Adder
11  PORT(
12  A : IN std_logic_vector(3 downto 0);
13  B : IN std_logic_vector(3 downto 0);
14  Cin : IN std_logic;
15  S : OUT std_logic_vector(3 downto 0);
16  Cout : OUT std_logic
17  );
18  END COMPONENT;
19
20  --Inputs
21  signal A : std_logic_vector(3 downto 0) := "0000" ;
22  signal B : std_logic_vector(3 downto 0) := "0000";
23  signal Cin : std_logic := '0';
24
25  --Outputs
26  signal S : std_logic_vector(3 downto 0);
27  signal Cout : std_logic;
28
29  BEGIN
30
31  -- Instantiate the Unit Under Test (UUT)
32  uut: Ripple_Adder PORT MAP (
33  A => A, B => B, Cin => Cin, S => S, Cout => Cout
34  );
35
36  -- Stimulus process
37  stim_proc: process
38  begin
39  -- hold reset state for 10 ns.
40  wait for 10 ns;
41  A <= "0110";
42  B <= "1100";
43
44  wait for 10 ns;
45  A <= "1111";
46  B <= "1100";
47
48  wait for 10 ns;
49  A <= "0110";
50  B <= "0111";
51
52  wait for 10 ns;
53  A <= "0110";
54  B <= "1110";
55
56  wait for 100 ns;
57  A <= "1111";
58  B <= "1111";
59
60  wait;
61  end process;
62  END;
```

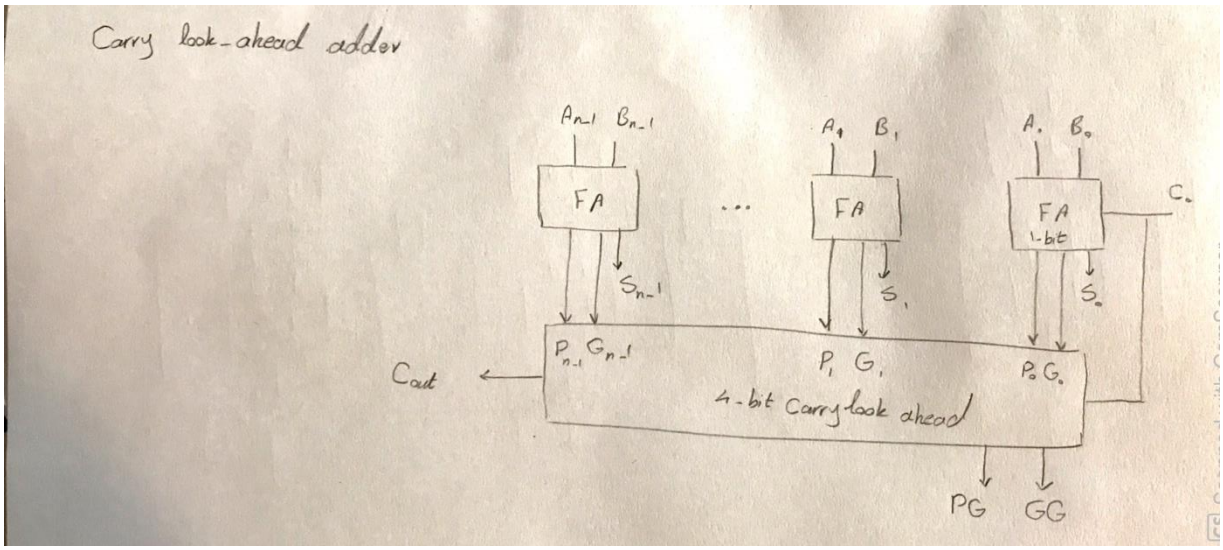
شکل 12: تست بنچ **Ripple adder**

Result of simulation in isim:



شکل 13 : **نتایج شبیه سازی Ripple adder**

ب) Carry_lookahead adder:



شکل 14 : **طراحی شماتیک Carry_lookahead adder**

delay(sum) = (2n/k +2)d

delay(carry) = (2n/k +1)d

cost = (n/k (3k +k(k+3)/2))g

Carry_lookahead adder به این صورت کار میکند که ابتدا Pi = Ai + Bi و Gi = Ai . Bi را بدست میاورد سپس با استفاده از فرمول Cn-1 = Gn-1 + Gn-2.Pn-1 + Gn-3.Pn-2.Pn-1 + … + Cin.P0.P1…Pn-1 میتوان مقدار Cout را بدست آورد و پس d زمان هم میتوان sum را با فرمول sum = Ai xor Bi xor Ci-1 بدست آورد.

behavioral of Carry_lookahead adder :

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Carry_Look_Ahead is
5  Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
6  B : in STD_LOGIC_VECTOR (3 downto 0);
7  Cin : in STD_LOGIC;
8  S : out STD_LOGIC_VECTOR (3 downto 0);
9  Cout : out STD_LOGIC);
10 end Carry_Look_Ahead;
11
12 architecture Behavioral of Carry_Look_Ahead is
13
14 component Partial_Full_Adder
15 Port ( A : in STD_LOGIC;
16 B : in STD_LOGIC;
17 Cin : in STD_LOGIC;
18 S : out STD_LOGIC;
19 P : out STD_LOGIC;
20 G : out STD_LOGIC);
21 end component;
22
23 signal c1,c2,c3: STD_LOGIC;
24 signal P,G: STD_LOGIC_VECTOR(3 downto 0);
25 begin
26
27 PFA1: Partial_Full_Adder port map( A(0), B(0), Cin, S(0), P(0), G(0));
28 PFA2: Partial_Full_Adder port map( A(1), B(1), c1, S(1), P(1), G(1));
29 PFA3: Partial_Full_Adder port map( A(2), B(2), c2, S(2), P(2), G(2));
30 PFA4: Partial_Full_Adder port map( A(3), B(3), c3, S(3), P(3), G(3));
31
32 c1 <= G(0) OR (P(0) AND Cin);
33 c2 <= G(1) OR (P(1) AND G(0)) OR (P(1) AND P(0) AND Cin);
34 c3 <= G(2) OR (P(2) AND G(1)) OR (P(2) AND P(1) AND G(0)) OR (P(2) AND P(1) AND P(0) AND Cin);
35 Cout <= G(3) OR (P(3) AND G(2)) OR (P(3) AND P(2) AND G(1)) OR (P(3) AND P(2) AND P(1) AND G(0)) OR (P(3) AND P(2) AND P(1) AND P(0) AND Cin);
36
37 end Behavioral;
38
```

شكل15 : توصيف **Carry_lookahead adder 4 bit**

behavioral of Partial full adder for Gi and Pi:

```
entity Partial_Full_Adder is
Port ( A : in STD_LOGIC;
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
P : out STD_LOGIC;
G : out STD_LOGIC);
end Partial_Full_Adder;

architecture Behavioral of Partial_Full_Adder is

begin

S <= A xor B xor Cin;
P <= A xor B;
G <= A and B;
```

شکل 16 : توصیف

Partial full adder

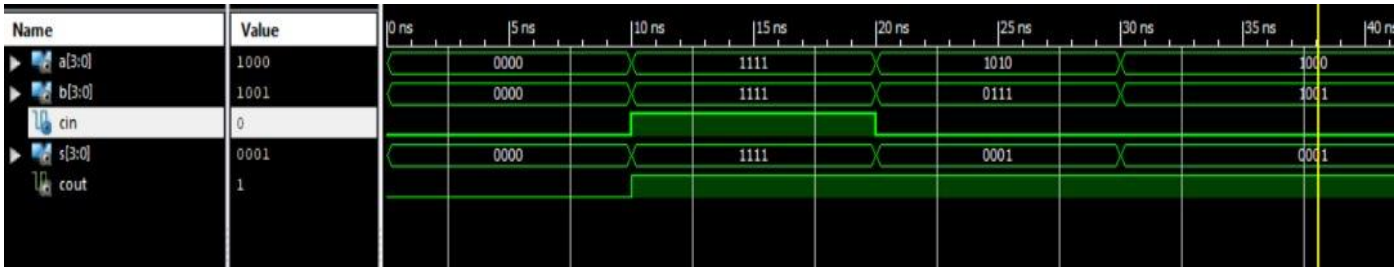
Test Bench:

```
COMPONENT Carry_Look_Ahead
PORT(
A : IN std_logic_vector(3 downto 0);
B : IN std_logic_vector(3 downto 0);
Cin : IN std_logic;
S : OUT std_logic_vector(3 downto 0);
Cout : OUT std_logic
);
END COMPONENT;

--Inputs
signal A : std_logic_vector(3 downto 0) := (others => '0');
signal B : std_logic_vector(3 downto 0) := (others => '0');
signal Cin : std_logic := '0';

--Outputs
signal S : std_logic_vector(3 downto 0);
signal Cout : std_logic;

BEGIN
 -- Instantiate the Unit Under Test (UUT)
uut: Carry_Look_Ahead PORT MAP (
A => A,
B => B,
Cin => Cin,
S => S,
Cout => Cout
);

-- Stimulus process
stim_proc: process
begin
-- hold reset state for 100 ns.
wait for 10 ns;
```

```
A <= "1111";
B <= "1111";
Cin <= '1';

wait for 10 ns;

A <= "1010";
B <= "0111";
Cin <= '0';

wait for 10 ns;

A <= "1000";
B <= "1001";
Cin <= '0';

wait;
end process;
END;
```

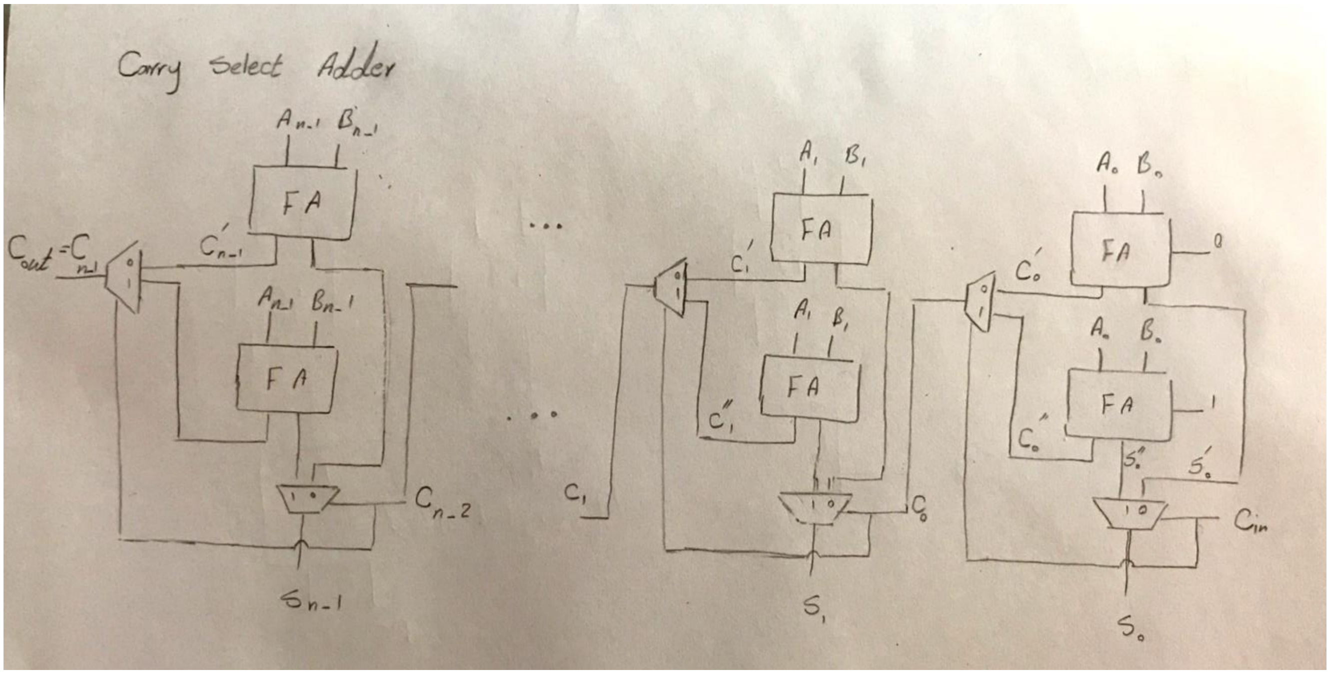
شکل 17 : تست بنچ **Carry_lookahead adder**

Result of simulation in isim:



شکل 18 : نتایج شبیه سازی **Carry_lookahead adder**

ج) Carry Select Adder:



شکل 19 : **طراحی شماتیک Carry select adder**

Carry select adder → delay(sum) = (2n +1)d MUX → delay = 3d

delay(carry) = (12n+2)d cost = (3k+1)g

cost = 13n+5

Carry select adder به این صورت کار میکند که ابتدا جفت بیت های A0 , B0 و A1, B1 و... An-1, Bn-1 جواب هر کدام یکبار با Cin = 0 و یک بار با Cin = 1 با یک FA بدست میاید تا از تاخیر بوجود آمده از Cout ها (که به عنوان Cin باید به ستون بعد منتقل شود) جلوگیری شود سپس با آمدن carry و با استفاده از Mux میتوان بدست آورد که کدام یک از جواب ها با Cin 0 یا 1 درست بوده و به همین ترتیب تا ستون آخر میرود.

behavioral of Carry select adder:

```
entity carry_select_adder is
Port (   X : in STD_LOGIC_VECTOR (3 downto 0);
   Y : in STD_LOGIC_VECTOR (3 downto 0);
   CARRY_IN : in STD_LOGIC;
   SUM : out STD_LOGIC_VECTOR (3 downto 0);
   CARRY_OUT : out STD_LOGIC);
end carry_select_adder;

architecture Behavioral of carry_select_adder is

component FA
Port (   A : in STD_LOGIC;
   B : in STD_LOGIC;
   Cin : in STD_LOGIC;
   S : out STD_LOGIC;
   Cout : out STD_LOGIC);
end component;

component mux2_1
port(
   A,B : in STD_LOGIC;
   Sel: in STD_LOGIC;
   Z: out STD_LOGIC
);
end component;

signal A,B,C0,C1: STD_LOGIC_VECTOR( 3 DOWNTO 0);
begin

FA1: FA PORT MAP(A => X(0),B => Y(0),Cin => '0' , S => A(0),Cout => C0(0));
FA2: FA PORT MAP(A => X(1),B => Y(1),Cin => C0(0) , S => A(1),Cout => C0(1));
FA3: FA PORT MAP(A => X(2),B => Y(2),Cin => C0(1) , S => A(2),Cout => C0(2));
FA4: FA PORT MAP(A => X(3),B => Y(3),Cin => C0(2) , S => A(3),Cout => C0(3));

FA5: FA PORT MAP(A => X(0),B => Y(0),Cin => '1' , S => B(0),Cout => C1(0));
FA6: FA PORT MAP(A => X(1),B => Y(1),Cin => C1(0) , S => B(1),Cout => C1(1));
FA7: FA PORT MAP(A => X(2),B => Y(2),Cin => C1(1) , S => B(2),Cout => C1(2));
FA8: FA PORT MAP(A => X(3),B => Y(3),Cin => C1(2) , S => B(3),Cout => C1(3));

MUX1: mux2_1 PORT MAP(A => A(0),B => B(0),Sel => CARRY_IN,Z => SUM(0));
MUX2: mux2_1 PORT MAP(A => A(1),B => B(1),Sel => CARRY_IN,Z => SUM(1));
MUX3: mux2_1 PORT MAP(A => A(2),B => B(2),Sel => CARRY_IN,Z => SUM(2));
MUX4: mux2_1 PORT MAP(A => A(3),B => B(3),Sel => CARRY_IN,Z => SUM(3));
MUX5: mux2_1 PORT MAP(A => C0(3),B =>C1(3),Sel => CARRY_IN,Z => CARRY_OUT);

end Behavioral;
```

شکل 20: توصیف **Carry select adder 4 bit با استفاده از FA و MUX**

Behavioral of Carry select adder:

```
port(

A,B : in STD_LOGIC;
Sel: in STD_LOGIC;
Z: out STD_LOGIC
);
end mux2_1;

architecture bhv of mux2_1 is
begin
process(A,B,Sel)
begin
if Sel = '0' then
Z <= A;
else
Z <= B;
end if;
```

شكل 21 : توصيف MUX

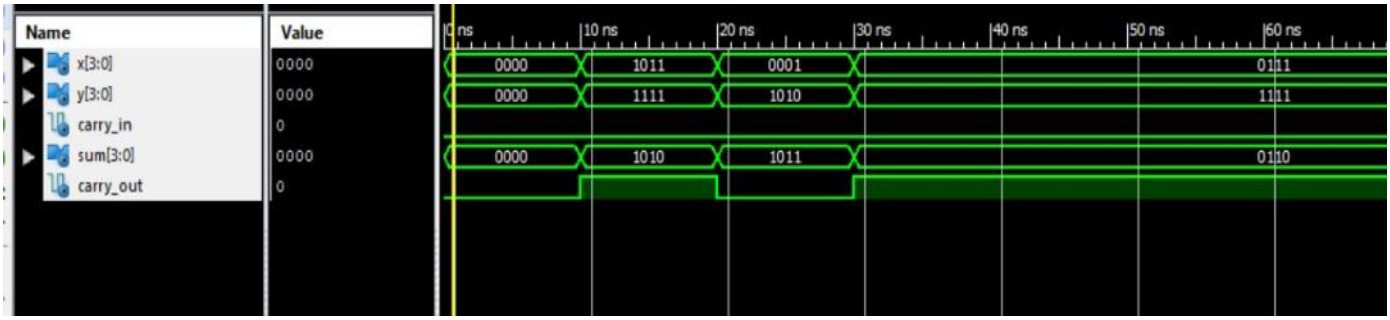
Test Bench:

```
COMPONENT carry_select_adder
PORT(
   X : IN std_logic_vector(3 downto 0);
   Y : IN std_logic_vector(3 downto 0);
   CARRY_IN : IN std_logic;
   SUM : OUT std_logic_vector(3 downto 0);
   CARRY_OUT : OUT std_logic
);
END COMPONENT;

--Inputs
signal X : std_logic_vector(3 downto 0) := "0000";
signal Y : std_logic_vector(3 downto 0) := "0000";
signal CARRY_IN : std_logic := '0';

--Outputs
signal SUM : std_logic_vector(3 downto 0);
signal CARRY_OUT : std_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: carry_select_adder PORT MAP (
X => X,
Y => Y,
CARRY_IN => CARRY_IN,
SUM => SUM,
CARRY_OUT => CARRY_OUT
);

 -- Stimulus process
stim_proc: process
begin
 -- hold reset state for 10 ns.
wait for 10 ns;
X <= "1011";
Y <= "1111";
```

```
49   wait for 10 ns;
50   X <= "0001";
51   Y <= "1010";
52
53   wait for 10 ns;
54   X <= "0111";
55   Y <= "1111";
56   wait;
57   end process;
58
59   END behavior;
```

شکل 22 : تست بنچ **Carry select adder**

Result of simulation in isim:



شکل 23 : نتایج شبیه سازی**Carry select adder**

نتیجه گیری و مقایسه سرعت جمع کننده ها :

توجه به delay و cost های بدست آمده در هر قسمت میتوان گفت:

۱) در Ripple adder سرعت بسیار پایین است چون باید منتظر carry قبلی باشد و این جمع کننده بهینه نیست.

۲)مقدار بهینگی در CLA به Kمورد نظر بستگی دارد زمانی که بخواهیم آن را به صورت آبشاری ببندیم که بدیهی است برای k=1 یا k=n مناسب نیست.

۳) در Carry select adder میتوان گفت با وجود cost بیشتر از دو برابر اما سرعت به خاطر از بین بردن مقدار زیادی از تاخیر های carry میتواند مناسب باشد و اگر به یک performance measure مناسب برسیم این جمع کننده گزینه خوبی است.