3η Εργαστηριακή Άσκηση στο μάθημα

Ψηφιακά Συστήματα VLSI

Ομάδα 13 :

Γρίβας Αριστοτέλης – el19889

Αυγουστής Μολτσάνοβ Εμίλ – el17064

Άσκηση 1 :

Ζητούμενο της άσκησης είναι η υλοποίηση ενός σύγχρονου πλήρη αθροιστή με περιγραφή συμπεριφοράς (Behavioral).

Ακολουθεί ο κώδικας που χρησιμοποιήθηκε για τη σχεδίαση του κυκλώματος :

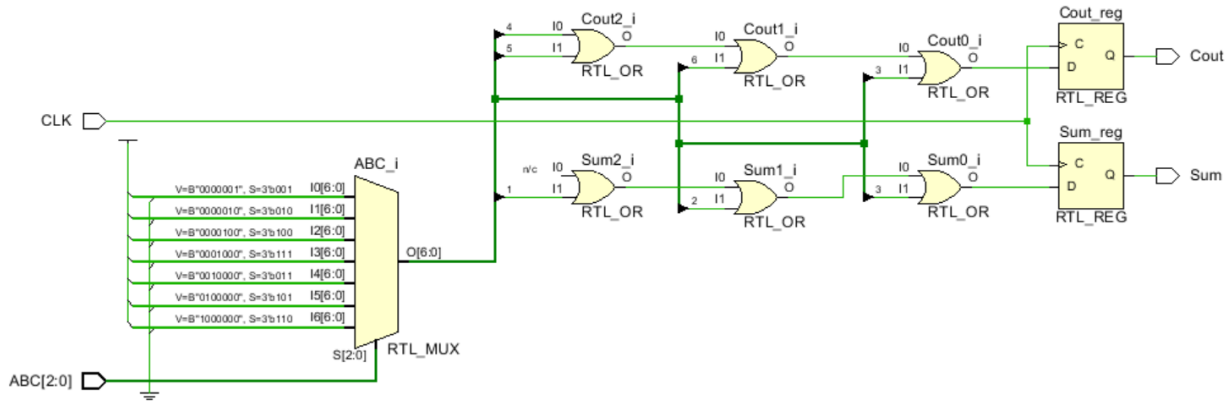
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity SynchronousFadd is
Port ( ABC : IN std_logic_vector(2 downto 0);
       CLK : IN std_logic;
      Cout,Sum : OUT std_logic
      );
end SynchronousFadd;

architecture Behavioral of SynchronousFadd is

begin

process (ABC,CLK)
begin

if (rising_edge(CLK)) then

    if (ABC = "001" or ABC = "010" or ABC = "100" or ABC =
"111") then
        Sum <= '1';
    else
        Sum <= '0';
    end if;

    if (ABC = "011" or ABC = "101" or ABC = "110" or ABC =
"111") then
```

```
        Cout <= '1';
    else
        Cout <= '0';
    end if;

end if;
end process;

end Behavioral;
```

Στον κώδικα αυτό περιγράφουμε πότε το άθροισμα και το κρατούμενο εξόδου γίνονται 1 ή 0 σύμφωνα με τις διάφορες τιμές των εισόδων A,B,C(ABC πίνακας 3 μεταβλητών).

Επιπλέον για να κάνουμε το κύκλωμα μας σύγχρονο έχουμε βάλει ως περιορισμό,οι έξοδοι να αλλάζουν μόνο στο θετικό παλμό του ρολογιού εισόδου CLK.

Παραθέτουμε και το rtl διάγραμμα :

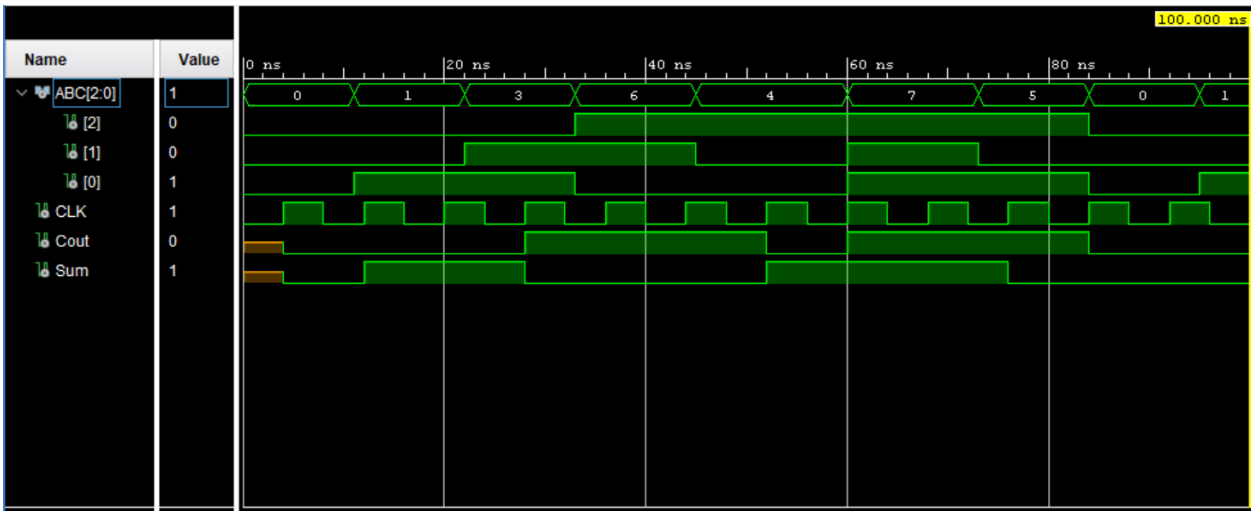


Παρατηρούμε ότι το κύκλωμα είναι ακριβώς όπως περιγράψαμε μέσω των πυλών OR του παραπάνω κώδικα,σε συνδυασμό με 2 D-flipflops τα οποία επιτρέπουν την αλλαγή των εξόδων μόνο στο θετικό παλμό του ρολογιού.Οι είσοδοι ABC δίνονται μέσω ενός πολυπλέκτη,ώστε να έχουμε όλους τους δυνατούς συνδιασμούς τιμών.

Το testebench που χρησιμοποιήθηκε για την επαλήθευση των αποτελεσμάτων δίνεται ως εξής :

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

entity SynchronousFadd_TB is
end SynchronousFadd_TB;

architecture Behavioral of SynchronousFadd_TB is

SIGNAL   ABC: std_logic_vector(2 DOWNTO 0) := "000";
SIGNAL   CLK: std_logic := '0';
SIGNAL   Cout: std_logic;
SIGNAL   Sum: std_logic;


begin
UUT : entity work.SynchronousFadd port map(ABC,CLK,Cout,Sum);

clk_process :process
   begin
        clk <= '0';
        wait for 4ns;
        clk <= '1';
        wait for 4ns;
   end process;

tb: PROCESS
begin
ABC <= "000";
wait for 11ns;
ABC <= "001";
wait for 11ns;
ABC <= "011";
wait for 11ns;
ABC <= "110";
wait for 12ns;
ABC <= "100";
wait for 15ns;
ABC <= "111";
wait for 13ns;
ABC <= "101";
wait for 11ns;
end process;

end Behavioral;
```

Και ακολουθούν οι κυματομορφές που παράγονται μέσω της προσομοίωσης :



Οι τιμές είναι οι θεωρητικά αναμενόμενες,με τις εξόδους να αλλάζουν σε κάθε κύκλο ρολογιού, παίρνοντας την επιθυμητή τιμή. Σημαντικό είναι το γεγονός πως η έξοδος του κυκλώματος είναι ορατή μετά από ένα αρχικό χρονικό διάστημα μισού κύκλου ρολογιού,καθώς έως τότε το ρολόι βρίσκεται χαμηλά,μη αποδίδοντας τιμή στην έξοδο.

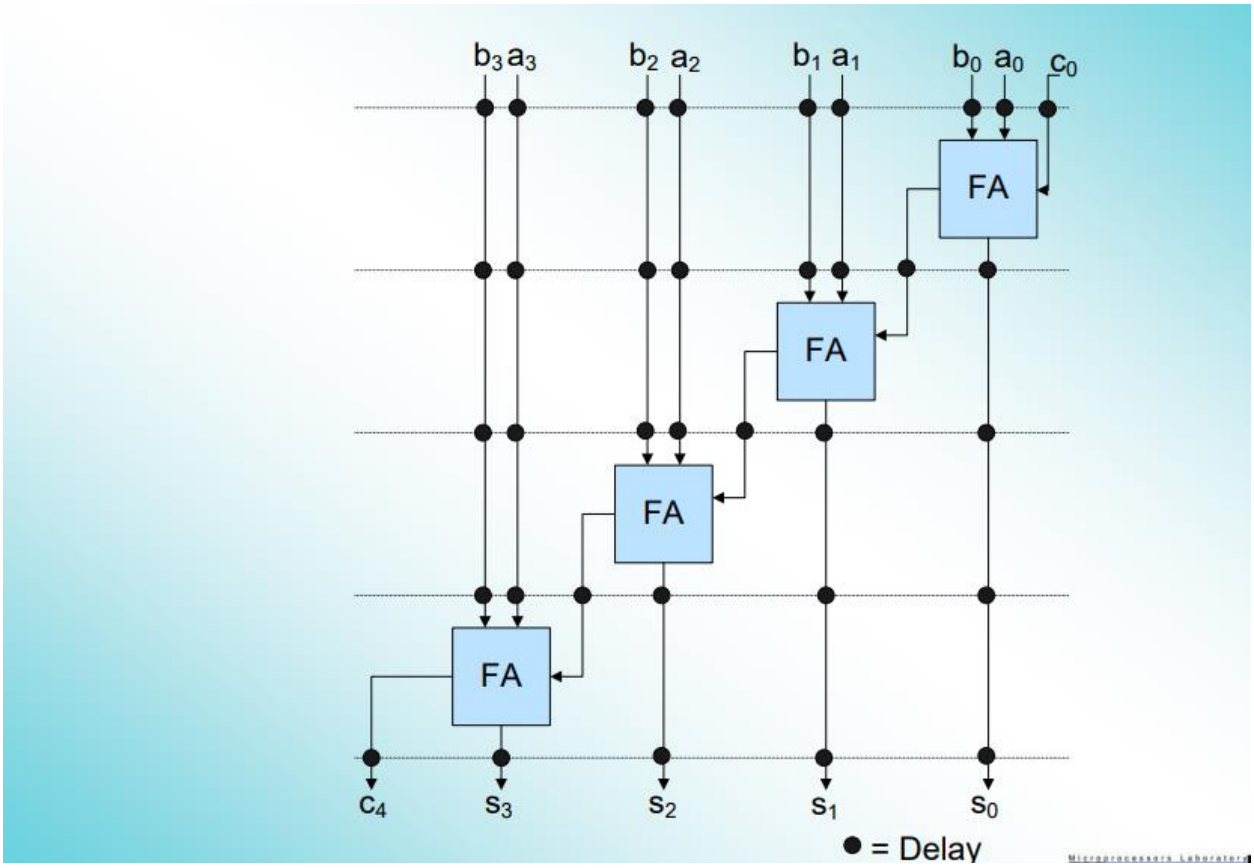
Τέλος,έχοντας τρέξει τη σύνθεση του κυκλώματος,μπορούμε να δούμε τα critical paths του κυκλώματος :

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	2	1	Cout_reg/C	Cout	4.076	3.276	0.800
Path 2	∞	2	2	1	Sum_reg/C	Sum	4.076	3.276	0.800
Path 3	∞	2	3	2	ABC[2]	Sum_reg/D	1.932	1.132	0.800
Path 4	∞	2	3	2	ABC[1]	Cout_reg/D	1.906	1.106	0.800

όπου οι τιμές των καθυστερήσεων είναι εντός των αναμενόμενων ορίων,ωστόσο δεν δίνονται πάντα από είσοδο σε έξοδο,αλλά από κάποιο ενδιάμεσο σημείο του κυκλώματος σε έξοδο.

Άσκηση 2 :

Ζητούμενο της άσκησης είναι η επέκταση του προηγούμενου ζητήματος έτσι ώστε να κατασκευαστεί ένας σύγχρονος αθροιστής διάδοσης κρατουμένου των 4 bits με χρήση της τεχνικής Pipeline. Για τον σχεδιασμό αυτού του κυκλώματος θα βασιστούμε στο παρακάτω σχηματικό διάγραμμα :



Όπως φαίνεται παραπάνω το κύκλωμα μας βασίζεται στην εν παραλλήλω σύνδεση 4 σύγχρονων αθροιστών(οι οποίοι υλοποιήθηκαν στην 1η άσκηση) με

το κρατούμενο εξόδου του κάθε ενός να αποτελεί το κρατούμενο εισόδου του επόμενου. Ωστόσο, παρατηρούμε ότι ανάμεσα από κάθε πλήρη αθροιστή πρέπει να προσθέσουμε μία καθυστέρηση,στην περίπτωσή μας ένα D-flipflop,πριν μεταφερθεί το κρατούμενο στον επόμενο αθροιστή. Επιπλέον,θέλουμε το αποτέλεσμά μας να παρέχεται όλο μαζί,δηλαδή και τα 5 bits εξόδου,συνεπώς είναι απαραίτητο να προσθέσουμε και τα αντίστοιχα flipflops για κάθε ένα από τα αθροίσματα. Τέλος, δεν πρέπει να ξεχνάμε τις καθυστερήσεις για τις εισόδους των αθροιστών. Επομένως :

Για το Sum(0) : θα χρειαστούμε 3 καθυστερήσεις (flipflops) αφού βγει από τον FADD1

Για το Sum(1) : θα χρειαστούμε 2 καθυστερήσεις (flipflops) αφού βγει από τον FADD2

Για το Sum(2) : θα χρειαστούμε 1 καθυστέρηση (flipflop) αφού βγει από τον FADD3

Για το Sum(3) : δεν θα χρειαστούμε καθυστερήσεις (flipflops) αφού βγει από τον FADD4,με αποτέλεσμα να το έχουμε στην έξοδο μαζί με τα υπόλοιπα Sums καθώς και το Cout (Sum(4))

Ο κώδικας που χρησιμοποιήθηκε δίνεται παρακάτω:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FourBitAdder is
Port ( CLK : IN std_logic;
       A,B: IN std_logic_vector(3 downto 0);
       Cin: IN std_logic;
       Sum : OUT std_logic_vector(4 downto 0)
            );
end FourBitAdder;

architecture Structural of FourBitAdder is

signal sigA1: std_logic;
signal sigB1: std_logic;
signal sigA2: std_logic_vector(1 downto 0);
signal sigB2: std_logic_vector(1 downto 0);
signal sigA3: std_logic_vector(2 downto 0);
```

```
signal sigB3: std_logic_vector(2 downto 0);
signal carrysig: std_logic_vector(2 downto 0);
signal sigsum0: std_logic_vector(2 downto 0);
signal sigsum1: std_logic_vector(1 downto 0);
signal sigsum2: std_logic;

component SynchronousFadd
Port ( ABC : IN std_logic_vector(2 downto 0);
       CLK : IN std_logic;
      Cout,Sum : OUT std_logic
      );
end component;

component reg
Port ( D,CLK : IN std_logic;
       Q : OUT std_logic
     );
end component;

begin

fadd1: SynchronousFadd
port map( ABC(0) => Cin ,
          ABC(1) => B(0),
          ABC(2) => A(0),
          CLK => CLK ,
          Cout => carrysig(0) ,
          Sum => sigsum0(0) );


a1_reg1: reg port map(D => A(1), CLK => CLK, Q =>sigA1);
b1_reg1: reg port map(D => B(1), CLK => CLK, Q =>sigB1);
a2_reg1: reg port map(D => A(2), CLK => CLK, Q =>sigA2(0));
b2_reg1: reg port map(D => B(2), CLK => CLK, Q =>sigB2(0));
a3_reg1: reg port map(D => A(3), CLK => CLK, Q =>sigA3(0));
b3_reg1: reg port map(D => B(3), CLK => CLK, Q =>sigB3(0));

fadd2: SynchronousFadd
port map( ABC(0) => carrysig(0) ,
          ABC(1) => sigB1,
          ABC(2) => sigA1,
          CLK => CLK ,
          Cout => carrysig(1) ,
          Sum => sigsum1(0) );
```

```
sum0_reg1: reg port map(D => sigsum0(0), CLK => CLK, Q
=>sigsum0(1));
a2_reg2: reg port map(D => sigA2(0), CLK => CLK, Q =>sigA2(1));
b2_reg2: reg port map(D => sigB2(0), CLK => CLK, Q =>sigB2(1));
a3_reg2: reg port map(D => sigA3(0), CLK => CLK, Q =>sigA3(1));
b3_reg2: reg port map(D => sigB3(0), CLK => CLK, Q =>sigB3(1));


fadd3: SynchronousFadd
port map( ABC(0) => carrysig(1) ,
          ABC(1) => sigA2(1),
          ABC(2) => sigB2(1),
          CLK => CLK ,
          Cout => carrysig(2) ,
          Sum => sigsum2 );


sum0_reg2: reg port map(D => sigsum0(1), CLK => CLK, Q
=>sigsum0(2));
sum1_reg1: reg port map(D => sigsum1(0), CLK => CLK, Q
=>sigsum1(1));
a3_reg3: reg port map(D => sigA3(1), CLK => CLK, Q =>sigA3(2));
b3_reg3: reg port map(D => sigB3(1), CLK => CLK, Q =>sigB3(2));

fadd4: SynchronousFadd
port map( ABC(0) => carrysig(2) ,
          ABC(1) => sigA3(2),
          ABC(2) => sigB3(2),
          CLK => CLK ,
          Cout => sum(4) ,
          Sum => Sum(3) );

sum0_reg3: reg port map(D => sigsum0(2), CLK => CLK, Q
=>Sum(0));
sum1_reg2: reg port map(D => sigsum1(1), CLK => CLK, Q
=>Sum(1));
sum2_reg1: reg port map(D => sigsum2, CLK => CLK, Q =>Sum(2));

end Structural;
```

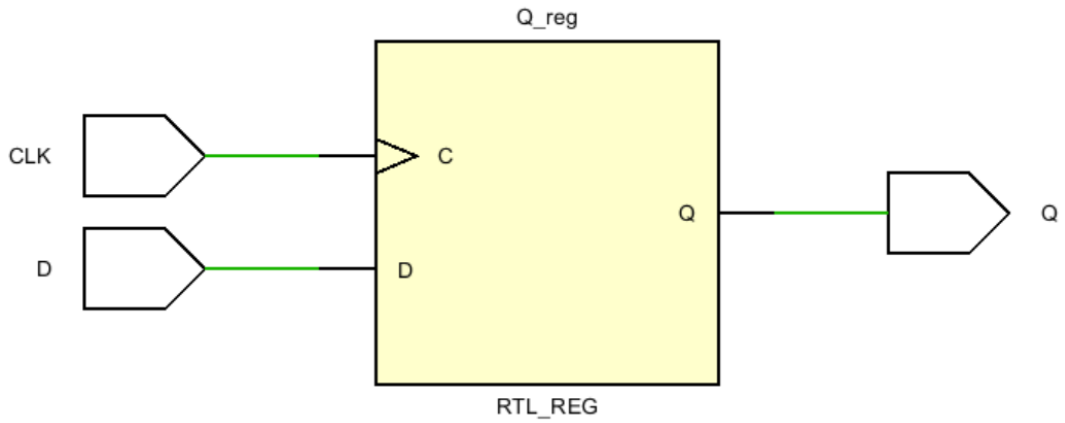
όπου γίνεται η χρήση του component SynchronousAdder 4 φορές, καθώς και του component reg που αποτελεί ένα D-flipflop .

Ο κώδικας που χρησιμοποιήθηκε για το D-flipflop παρέχεται παρακάτω :

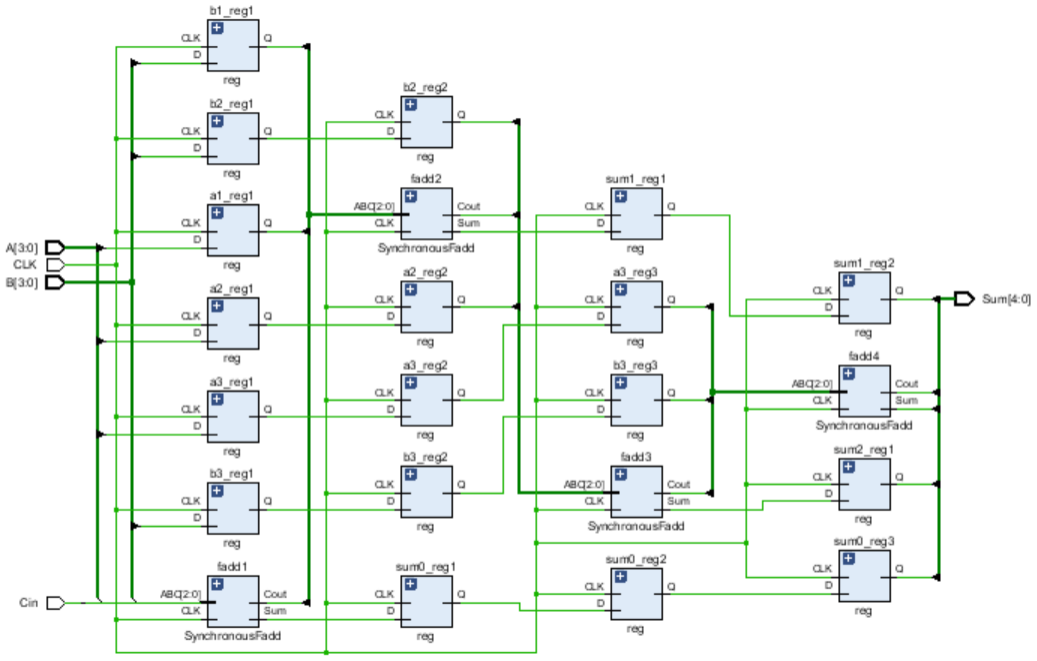
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg is
Port ( D,CLK : IN std_logic;
       Q : OUT std_logic
     );
end reg;

architecture behavioral of reg is
begin
    process(CLK)
    begin
        if(rising_edge(CLK)) then
            Q <= D;
        end if;
    end process;
end behavioral;
```

Τα rtl διαγράμματα που προκύπτουν είναι τα εξής :

Για τα flip flops(component reg):



Για το κύκλωμα του 4-bitAdder :



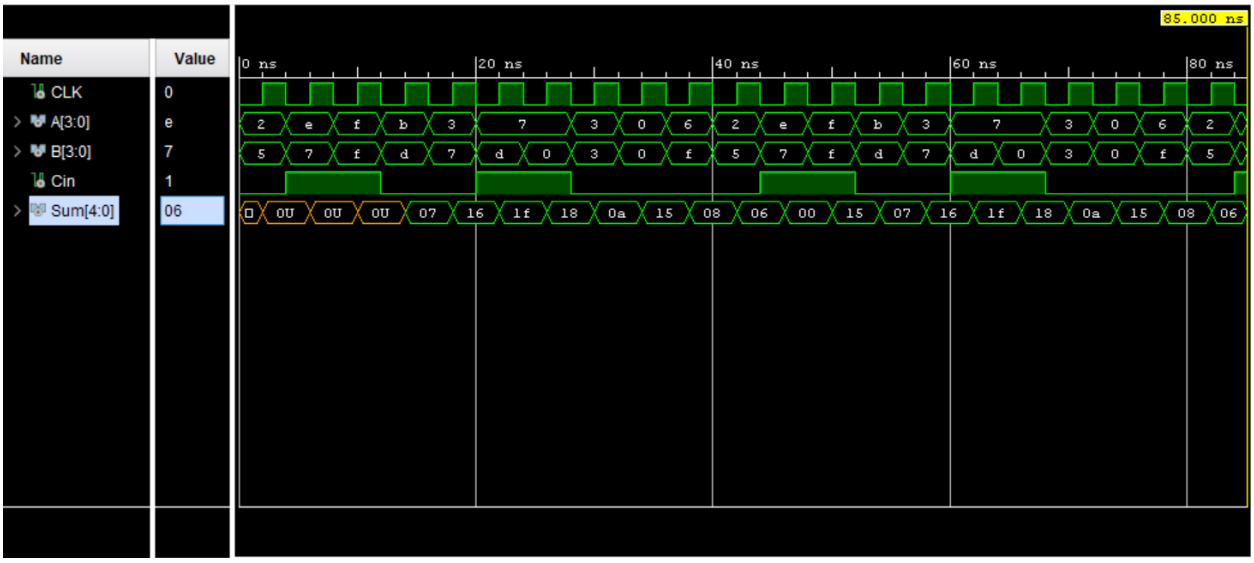
και είναι ακριβώς αυτό που περιμέναμε,4 SynchronousAdders παράλληλα,μαζί με τα αναγκαία ff.

Το testebench που χρησιμοποιήθηκε για την επαλήθευση των αποτελεσμάτων δίνεται ως εξής :

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

entity PipelinedFadd is
end PipelinedFadd;

architecture Behavioral of PipelinedFadd is

SIGNAL CLK : std_logic := '0';
SIGNAL  A,B : std_logic_vector(3 downto 0):= "0000";
SIGNAL Cin : std_logic := '0';
SIGNAL  Sum :  std_logic_vector(4 downto 0);
--SIGNAL  Coutp :  std_logic;


begin
UUT : entity work.FourBitAdder port map(CLK => CLK,A => A,B =>
B,Cin =>Cin,Sum=>Sum);


clk_process :process
   begin
        CLK <= '0';
        wait for 2ns;
        CLK <= '1';
        wait for 2ns;
   end process;

tb: PROCESS
begin
A <= "0010";
B <= "0101";
Cin <= '0';
wait for 4ns;
A <= "1110";
B <= "0111";
Cin <= '1';
wait for 4ns;
A <= "1111";
B <= "1111";
Cin <= '1';
wait for 4ns;
A <= "1010";
B <= "0101";
```

```
Cin <= '1';
A <= "1011";
B <= "1101";
Cin <= '0';
wait for 4ns;
A <= "0011";
B <= "0111";
Cin <= '0';
wait for 4ns;
A <= "0111";
B <= "1101";
Cin <= '1';
wait for 4ns;
A <= "0111";
B <= "0000";
Cin <= '1';
wait for 4ns;
A <= "0011";
B <= "0011";
Cin <= '0';
wait for 4ns;
A <= "0000";
B <= "0000";
Cin <= '0';
wait for 4ns;
A <= "0110";
B <= "1111";
Cin <= '0';
wait for 4ns;
end process;

end Behavioral;
```

Και ακολουθούν ο κυματομορφές που παράγονται μέσω της προσομοίωσης :



Οι τιμές είναι οι θεωρητικά αναμενόμενες,και το αποτέλεσμα δίνεται ορθά σε κάθε κύκλο ρολογιού,μετά απο καθυστέρηση Tlatency = 4 cycles.

Η καθυστέρηση αυτή είναι λογική καθώς το πρώτο αποτέλεσμα για να παραχθεί, πρέπει τα δεδομένα να περάσουν από όλα τα στάδια των flipflops για πρώτη φορά,ενώ τα επόμενα επεξεργάζονται παράλληλα με τα πρώτα.Συνεπώς η λειτουργία του κυκλώματος μας είναι η ορθή.

Τέλος,έχοντας τρέξει τη σύνθεση του κυκλώματος,μπορούμε να δούμε τα critical paths του κυκλώματος :

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	2	1	sum2_reg1/Q_reg/C	Sum[2]	4.076	3.276	0.800
Path 2	∞	2	2	1	fadd4/Sum_reg/C	Sum[3]	4.076	3.276	0.800
Path 3	∞	2	2	1	fadd4/Cout_reg/C	Sum[4]	4.076	3.276	0.800
Path 4	∞	2	2	1	sum0_reg3/Q_reg/C	Sum[0]	4.058	3.258	0.800
Path 5	∞	2	2	1	sum1_reg2/Q_reg/C	Sum[1]	4.058	3.258	0.800
Path 6	∞	2	3	2	B[0]	sum0_reg2/Q_reg_srl3/D	2.265	1.132	1.133
Path 7	∞	2	3	2	A[0]	fadd1/Cout_reg/D	1.906	1.106	0.800
Path 8	∞	2	2	2	a1_reg1/Q_reg/C	sum1_reg1/Q_reg_srl2/D	1.862	0.777	1.085
Path 9	∞	1	2	1	A[1]	a1_reg1/Q_reg/D	1.782	0.982	0.800
Path 10	∞	1	2	1	A[2]	a2_reg1/Q_reg/D	1.782	0.982	0.800

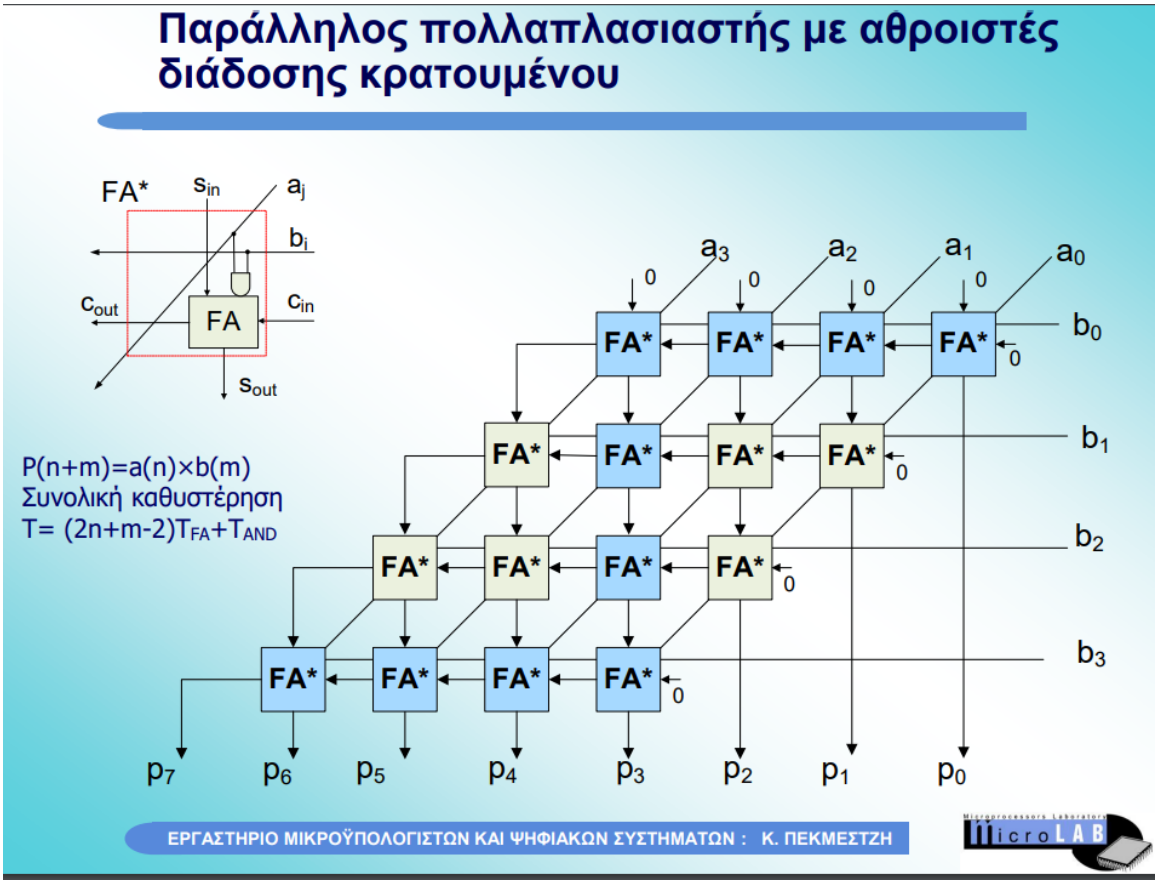
Παρατηρούμε όπως και πριν πως τα πιο κρίσιμα μονοπάτια δεν δίνονται από είσοδο σε έξοδο,αλλά από κάποιο ενδιάμεσο σημείο του κυκλώματος έως την

έξοδο.Αυτό οφείλεται πιθανότατα στη σύνθεση που πραγματοποιεί το πρόγραμμα για την πλακέτα της επιλογής μας(zybo),και παρατηρούμε πως σε σύγκριση με την άσκηση 3 της προηγούμενης εργαστηριακής άσκησης οι καθυστερήσεις των κρίσιμων μονοπατιών είναι μικρότερες. Αυτό δεν μπορεί να είναι ορθό επειδή στο τωρινό μας 4bit adder σε σύγκριση με αυτόν της $2^{ης}$ εργαστηριακής άσκησης μεσολαβούν περισσότερα κυκλώματα μεταξύ της κάθε εισόδου και της κάθε εξόδου, οπότε το critical path πρέπει να προκύπτει μεγαλύτερο.

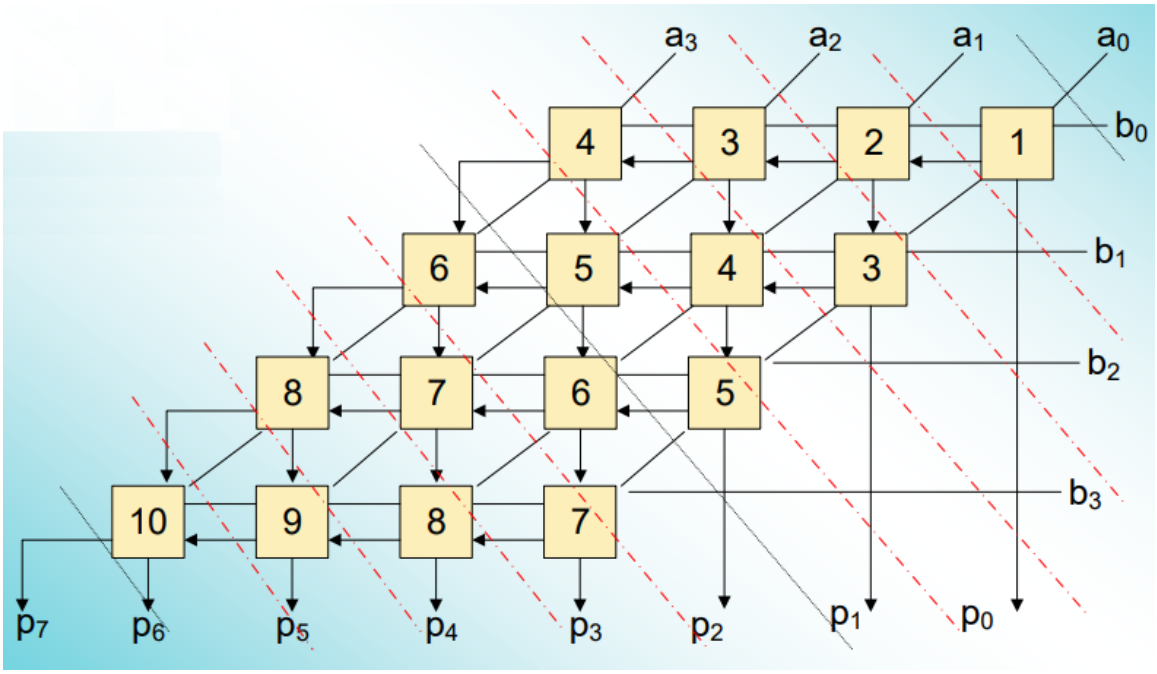
Επειδή στο κύκλωμα μας από τις εισόδους μέχρι τις εξόδους μεσολαβούν 4 επίπεδα καθυστερήσεων (D flip flops), εκτιμούμε πως το ορθό critical path ισούται με 4 clock cycles. Δηλάδη δεν είναι σταθερό, αλλά εξαρτάται από την περίοδο του ρολογιού.

Άσκηση 3 :

Προκειμένου να φτιάξουμε τον συστολικό πολλαπλασιαστή διάδοσης κρατουμένου των 4 bits, παίρνουμε τον ακόλουθο παράλληλο πολλαπλασιαστή διάδοσης κρατουμένου:



Και εισάγουμε καθυστερήσεις (d flip flops) όπου χρειαστεί, σύμφωνα με το παρακάτω σχήμα.



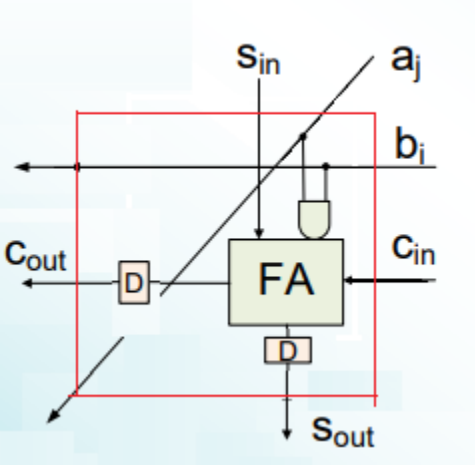
Οι καθυστερήσεις εισάγονται με την ακόλουθη λογική:

- Θέλουμε στον <u>κάθε</u> FA* να παρουσιάζονται σε <u>κάθε</u> παλμό ρολογιού σωστές τιμές εισόδων, οπότε χρειάζεται να φτάνουν με την ίδια καθυστέρηση.
- Θέλουμε το αποτέλεσμα να παρέχεται όλο μαζί.

Για την υλοποίηση του pipelined multiplier, αρχικά φτιάχνουμε τις ακόλουθες δομές, που έπειτα θα χρησιμοποιηθούν ως components:

- mul_synch_fa
- dff
- dff_two
- dff_three
- dff_five
- dff_seven
- dff_nine

1) **mul_synch_fa**



Αυτή η δομή κατασκευάζεται προκειμένου να γίνει χρήση του σύγχρονου πλήρη αθροιστή, όπως ζητάει η εκφώνηση.

Κώδικας VHDL

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

entity mul_synch_fa is
port(
A,B,S_in,clk,C_in : in std_logic;
C_out, S_out: out std_logic);
end mul_synch_fa;

architecture Behavioral of mul_synch_fa is

component synch_fa is
port(
A,B,Cin,clk : in std_logic;
Sum, Cout : out std_logic);
```

```
end component;

signal s1 : std_logic;

begin

s1 <= A and B;

m1: synch_fa port map( A => S_in, B => s1, Cin => C_in, Sum => S_out, Cout => C_out, clk =>
clk);

end Behavioral;
```

Παρατήρηση: το component synch_fa είναι ο σύγχρονος πλήρης αθροιστής του 1ου ερωτήματος.

2) Dff

Είναι απλώς ένα d flip flop.

Κώδικας VHDL

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

entity dff is
Port (
D,clock : in std_logic;
Q : out std_logic);
end dff;
```

```
architecture behavioral of dff is
begin
   process(clock)
   begin
      if(rising_edge(clock)) then
         Q <= D;
      end if;
   end process;
end behavioral;
```

3) Dff_two

Είναι δύο d flip flops εν σειρά.

Κώδικας VHDL

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

entity dff_two is
port(
D,clk   : in std_logic;
Q       : out std_logic);
end dff_two;

architecture Behavioral of dff_two is

component dff is
```

```
Port (
D,clock : in std_logic;
Q : out std_logic);
end component;

signal s0 : std_logic;

begin

d1: dff port map( D => D   , Q => s0, clock => clk);
d2: dff port map( D => s0, Q => Q, clock => clk);



end Behavioral;
```

4) Dff_three

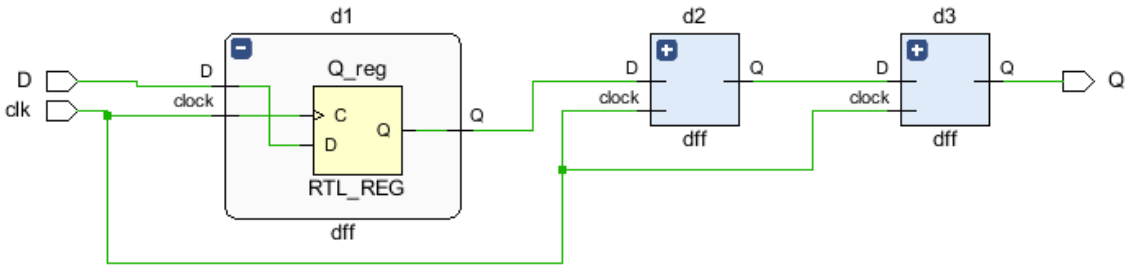
Είναι τρία d flip flops εν σειρά.

Κώδικας VHDL

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

entity dff_three is
port(
D,clk   : in std_logic;
Q       : out std_logic);
end dff_three;
```

```
architecture Behavioral of dff_three is

component dff is
Port (
D,clock : in std_logic;
Q : out std_logic);
end component;

signal s : std_logic_vector(1 downto 0);

begin

d1: dff port map( D => D   , Q => s(0), clock => clk);
d2: dff port map( D => s(0), Q => s(1), clock => clk);
d3: dff port map( D => s(1), Q => Q   , clock => clk);


end Behavioral;
```

Παραθέτουμε και το rtl schematic για σαφήνεια.



5) Dff_five

Είναι πέντε d flip flop εν σειρά.

6) **Dff_seven**

Είναι εφτά d flip flop εν σειρά.

7) **Dff_nine**

Είναι εννέα d flip flop εν σειρά.

Οι κώδικες vhdl των dff_five, dff_seven, dff_nine αποτέλουν ευθύγραμμη επέκταση του κώδικα vhdl του dff_three, οπότε δεν παρουσιάζονται.

Pipelined Multiplier

Κώδικας VHDL

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

entity pipelined_multiplier is
port(
clk : in std_logic;
A,B : in std_logic_vector(3 downto 0);
P  : out std_logic_vector(7 downto 0));
end pipelined_multiplier;

architecture Behavioral of pipelined_multiplier is

component dff is
Port (
D,clock : in std_logic;
Q : out std_logic);
end component;
```

```
component dff_two is
port(
D,clk   : in std_logic;
Q       : out std_logic);
end component;

component dff_three is
port(
D,clk   : in std_logic;
Q       : out std_logic);
end component;

component dff_five is
port(
D,clk   : in std_logic;
Q       : out std_logic);
end component;

component dff_seven is
port(
D,clk   : in std_logic;
Q       : out std_logic);
end component;

component dff_nine is
port(
D,clk   : in std_logic;
```

```
Q      : out std_logic);
end component;

component mul_synch_fa is
port(
A,B,S_in,clk,C_in : in std_logic;
C_out, S_out: out std_logic);
end component;

signal s_a0,s_a1,s_a2,s_a3: std_logic_vector(6 downto 0);
signal s_a3_00 : std_logic;

signal s_b0: std_logic_vector(3 downto 0);
signal s_b1: std_logic_vector(5 downto 2);
signal s_b2: std_logic_vector(7 downto 4);
signal s_b3: std_logic_vector(9 downto 6);

signal s_cout_hor0,s_cout_hor1,s_cout_hor2,s_cout_hor3: std_logic_vector(3 downto 0);

signal s_sout_ver0,s_sout_ver6 : std_logic;
signal s_sout_ver1 : std_logic_vector(1 downto 0);
signal s_sout_ver2,s_sout_ver5 : std_logic_vector(2 downto 0);
signal s_sout_ver3, s_sout_ver4 : std_logic_vector(3 downto 0);



begin
```

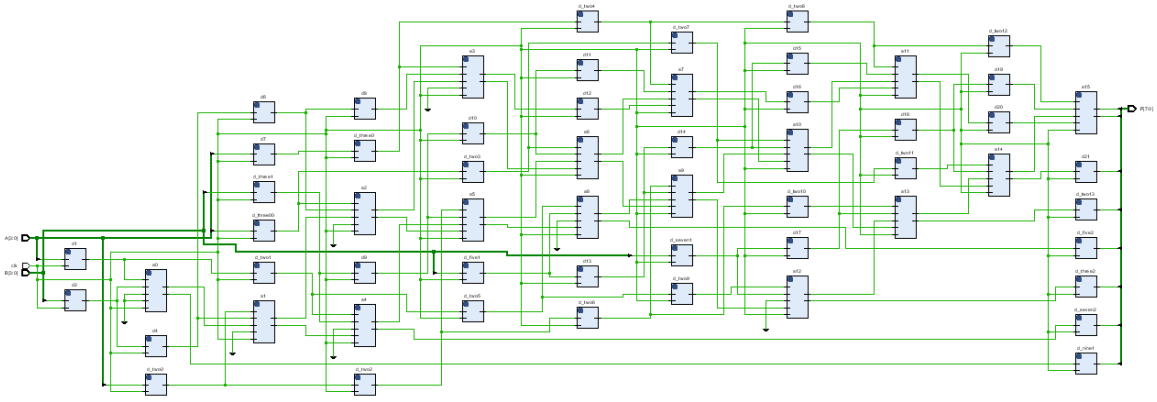
```
--Horizontal level 0

d1: dff port map( D => A(0), clock => clk, Q => s_a0(0) );

d2: dff port map( D => B(0), clock => clk, Q => s_b0(0) );

a0: mul_synch_fa port map( A => s_a0(0), B => s_b0(0), S_in =>'0', C_in =>'0', S_out =>
s_sout_ver0, C_out => s_cout_hor0(0), clk => clk );


d_two0: dff_two port map( D => A(1), clk => clk, Q => s_a1(0) );

d4: dff port map( D => s_b0(0), clock => clk, Q => s_b0(1) );

a1: mul_synch_fa port map( A => s_a1(0), B => s_b0(1), S_in =>'0', C_in => s_cout_hor0(0),
S_out => s_sout_ver1(0), C_out => s_cout_hor0(1), clk => clk );


d_three00: dff_three port map( D => A(2), clk => clk, Q => s_a2(0) );

d6: dff port map( D => s_b0(1), clock => clk, Q => s_b0(2) );

a2: mul_synch_fa port map( A => s_a2(0), B => s_b0(2), S_in =>'0', C_in => s_cout_hor0(1),
S_out => s_sout_ver2(0), C_out => s_cout_hor0(2), clk => clk );


d7: dff port map( D => A(3), clock => clk, Q => s_a3_00 );

d_three0: dff_three port map( D => s_a3_00, clk => clk, Q => s_a3(0) );

d8: dff port map( D => s_b0(2), clock => clk, Q => s_b0(3) );

a3: mul_synch_fa port map( A => s_a3(0), B => s_b0(3), S_in =>'0', C_in => s_cout_hor0(2),
S_out => s_sout_ver3(0), C_out => s_cout_hor0(3), clk => clk );

--


--Horizontal level 1

d_two1: dff_two port map( D => s_a0(0), clk => clk, Q => s_a0(2) );

d_three1: dff_three port map ( D => B(1), clk => clk, Q => s_b1(2) );

a4: mul_synch_fa port map( A => s_a0(2), B => s_b1(2), S_in => s_sout_ver1(0), C_in => '0',
S_out => s_sout_ver1(1), C_out => s_cout_hor1(0), clk => clk );


d_two2: dff_two port map( D => s_a1(0), clk => clk, Q => s_a1(2) );
```

```
d9: dff port map( D => s_b1(2), clock => clk, Q => s_b1(3) );

a5: mul_synch_fa port map( A => s_a1(2), B => s_b1(3), S_in => s_sout_ver2(0), C_in =>
s_cout_hor1(0), S_out => s_sout_ver2(1), C_out => s_cout_hor1(1), clk => clk );


d_two3: dff_two port map( D => s_a2(0), clk => clk, Q => s_a2(2) );

d10: dff port map( D => s_b1(3), clock => clk, Q => s_b1(4) );

a6: mul_synch_fa port map( A => s_a2(2), B => s_b1(4), S_in => s_sout_ver3(0), C_in =>
s_cout_hor1(1), S_out => s_sout_ver3(1), C_out => s_cout_hor1(2), clk => clk );


d_two4: dff_two port map( D => s_a3(0), clk => clk, Q => s_a3(2) );

d11: dff port map( D => s_b1(4), clock => clk, Q => s_b1(5) );

d12: dff port map( D => s_cout_hor0(3), clock => clk, Q => s_sout_ver4(0) );

a7: mul_synch_fa port map( A => s_a3(2), B => s_b1(5), S_in => s_sout_ver4(0), C_in =>
s_cout_hor1(2), S_out => s_sout_ver4(1), C_out => s_cout_hor1(3), clk => clk );


--Horizontal level 2

d_two5: dff_two port map( D => s_a0(2), clk => clk, Q => s_a0(4) );

d_five1: dff_five port map ( D => B(2), clk => clk, Q => s_b2(4) );

a8: mul_synch_fa port map( A => s_a0(4), B => s_b2(4), S_in => s_sout_ver2(1), C_in => '0',
S_out => s_sout_ver2(2), C_out => s_cout_hor2(0), clk => clk );


d_two6: dff_two port map( D => s_a1(2), clk => clk, Q => s_a1(4) );

d13: dff port map( D => s_b2(4), clock => clk, Q => s_b2(5) );

a9: mul_synch_fa port map( A => s_a1(4), B => s_b2(5), S_in => s_sout_ver3(1), C_in =>
s_cout_hor2(0), S_out => s_sout_ver3(2), C_out => s_cout_hor2(1), clk => clk );


d_two7: dff_two port map( D => s_a2(2), clk => clk, Q => s_a2(4) );

d14: dff port map( D => s_b2(5), clock => clk, Q => s_b2(6) );

a10: mul_synch_fa port map( A => s_a2(4), B => s_b2(6), S_in => s_sout_ver4(1), C_in =>
s_cout_hor2(1), S_out => s_sout_ver4(2), C_out => s_cout_hor2(2), clk => clk );
```

```
d_two8: dff_two port map( D => s_a3(2), clk => clk, Q => s_a3(4) );

d15: dff port map( D => s_b2(6), clock => clk, Q => s_b2(7) );

d16: dff port map( D => s_cout_hor1(3), clock => clk, Q => s_sout_ver5(0) );

a11: mul_synch_fa port map( A => s_a3(4), B => s_b2(7), S_in => s_sout_ver5(0), C_in =>
s_cout_hor2(2), S_out => s_sout_ver5(1), C_out => s_cout_hor2(3), clk => clk );


--Horizontal level 3

d_two9: dff_two port map( D => s_a0(4), clk => clk, Q => s_a0(6) );

d_seven1: dff_seven port map ( D => B(3), clk => clk, Q => s_b3(6) );

a12: mul_synch_fa port map( A => s_a0(6), B => s_b3(6), S_in => s_sout_ver3(2), C_in => '0',
S_out => s_sout_ver3(3), C_out => s_cout_hor3(0), clk => clk );


d_two10: dff_two port map( D => s_a1(4), clk => clk, Q => s_a1(6) );

d17: dff port map( D => s_b3(6), clock => clk, Q => s_b3(7) );

a13: mul_synch_fa port map( A => s_a1(6), B => s_b3(7), S_in => s_sout_ver4(2), C_in =>
s_cout_hor3(0), S_out => s_sout_ver4(3), C_out => s_cout_hor3(1), clk => clk );


d_two11: dff_two port map( D => s_a2(4), clk => clk, Q => s_a2(6) );

d18: dff port map( D => s_b3(7), clock => clk, Q => s_b3(8) );

a14: mul_synch_fa port map( A => s_a2(6), B => s_b3(8), S_in => s_sout_ver5(1), C_in =>
s_cout_hor3(1), S_out => s_sout_ver5(2), C_out => s_cout_hor3(2), clk => clk );


d_two12: dff_two port map( D => s_a3(4), clk => clk, Q => s_a3(6) );

d19: dff port map( D => s_b3(8), clock => clk, Q => s_b3(9) );

d20: dff port map( D => s_cout_hor2(3), clock => clk, Q => s_sout_ver6 );

a15: mul_synch_fa port map( A => s_a3(6), B => s_b3(9), S_in => s_sout_ver6, C_in =>
s_cout_hor3(2), S_out => P(6), C_out => P(7), clk => clk );

--


d21: dff port map( D => s_sout_ver5(2), clock => clk, Q => P(5) );
```

```
d_two13: dff_two port map ( D => s_sout_ver4(3), clk => clk, Q => P(4) );
d_three2: dff_three port map( D => s_sout_ver3(3), clk => clk, Q => P(3) );
d_five2: dff_five port map( D => s_sout_ver2(2), clk => clk, Q => P(2) );
d_seven2: dff_seven port map( D => s_sout_ver1(1), clk => clk, Q => P(1) );
d_nine1: dff_nine port map( D => s_sout_ver0, clk => clk, Q => P(0) );

end Behavioral;
```

Rtl schematic



Testbench

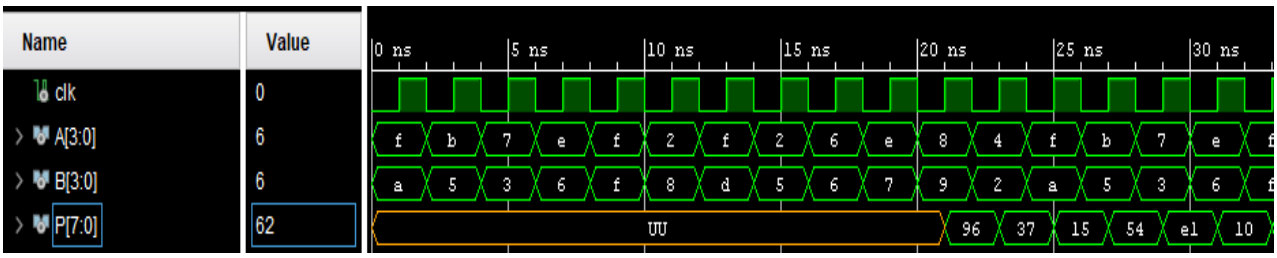
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_pipelined_multiplier is
end tb_pipelined_multiplier;

architecture Behavioral of tb_pipelined_multiplier is

component pipelined_multiplier is
```

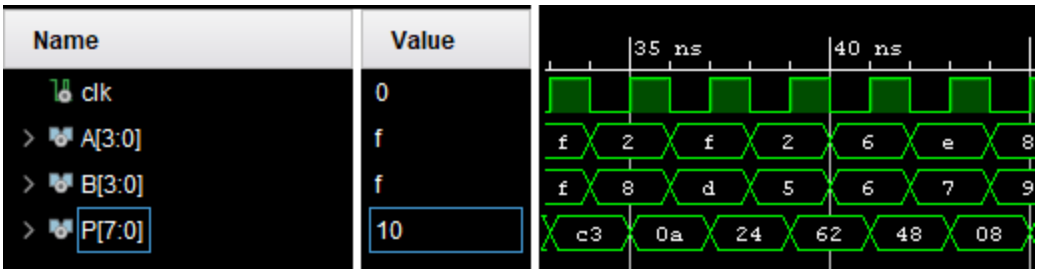
```
port(
clk : in std_logic;
A,B : in std_logic_vector(3 downto 0);
P  : out std_logic_vector(7 downto 0));
end component;

signal clk : std_logic :='0';
signal A,B : std_logic_vector(3 downto 0) :="0000";
signal P   : std_logic_vector(7 downto 0) :="00000000";

begin

tb: pipelined_multiplier port map( A => A, B => B, P => P, clk => clk );

 clk_proc :process is
 begin
 clk <= '0';
 wait for 1ns;
 clk <= '1';
 wait for 1ns;

 end process;

 proc: process is
 begin

 A <= "1111";
 B <= "1010";
```

```
wait for 2ns;

A <= "1011";
B <= "0101";
wait for 2ns;

A <= "0111";
B <= "0011";
wait for 2ns;

A <= "1110";
B <= "0110";
wait for 2ns;

A <= "1111";
B <= "1111";
wait for 2ns;

A <= "0010";
B <= "1000";
wait for 2ns;

A <= "1111";
B <= "1101";
wait for 2ns;

A <= "0010";
B <= "0101";
```

```
  wait for 2ns;

  A <= "0110";
  B <= "0110";
  wait for 2ns;

  A <= "1110";
  B <= "0111";
  wait for 2ns;

  A <= "1000";
  B <= "1001";
  wait for 2ns;

  A <= "0100";
  B <= "0010";
  wait for 2ns;

  end process proc;

end Behavioral;
```

Simulation





Παρατηρούμε ότι μετα από καθυστέρηση T(Latency)=11 clock cycles, το κύκλωμα παράγει σωστά αποτελέσματα σε κάθε παλμό ρολογιού. Η καθυστέρηση T(Latency) είναι λογικό να ισούται με 11 περιόδους του ρολογιού μιας και απο τις εισόδους μέχρι τις εξόδους μεσολαβούν 11 επίπεδα d flip flop.

Critical path:

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	2	1	d21/Q_reg/C	P[5]	4.076	3.276	0.800
Path 2	∞	2	2	1	a15/m1/Sum_reg/C	P[6]	4.076	3.276	0.800
Path 3	∞	2	2	1	a15/m1/Cout_reg/C	P[7]	4.076	3.276	0.800
Path 4	∞	2	2	1	d_nine1/d9/Q_reg/C	P[0]	4.058	3.258	0.800
Path 5	∞	2	2	1	d_seven2/d7/Q_reg/C	P[1]	4.058	3.258	0.800
Path 6	∞	2	2	1	d_five2/d5/Q_reg/C	P[2]	4.058	3.258	0.800

Παρατηρούμε όπως και πριν πως τα πιο κρίσιμα μονοπάτια δεν δίνονται από είσοδο σε έξοδο,αλλά από κάποιο ενδιάμεσο σημείο του κυκλώματος έως την έξοδο.Αυτό οφείλεται πιθανότατα στη σύνθεση που πραγματοποιεί το πρόγραμμα για την πλακέτα της επιλογής μας(zybo). Πιστεύουμε πως αυτό δεν critical path δεν είναι ορθό.

Επειδή στο κύκλωμα μας από τις εισόδους μέχρι τις εξόδους μεσολαβούν 11 επίπεδα καθυστερήσεων (D flip flops), εκτιμούμε πως το ορθό critical path ισούται με 11 clock cycles. Δηλάδη δεν είναι σταθερό, αλλά εξαρτάται από την περίοδο του ρολογιού.