2η Εργαστηριακή Άσκηση στο μάθημα

Ψηφιακά Συστήματα VLSI

Ομάδα 13 :

Γρίβας Αριστοτέλης – el19889

Αυγουστής Μολτσάνοβ Εμίλ – el17064

Άσκηση 1: Half adder

A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Με βάση τον παραπάνω πίνακα αληθείας:

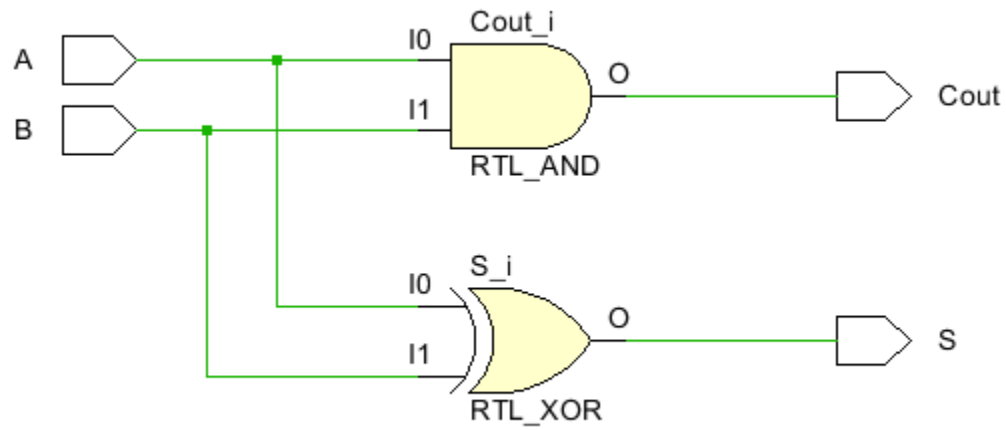
S=A'B+AB'=xor(A,B) , Cout=AB

Κώδικας VHDL:

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

entity HalfAdder is
port(
A,B : in std_logic;
Cout,S: out std_logic);
end HalfAdder;

architecture Behavioral of Halfadder is
```

```
begin
S <= A xor B;
Cout <= A and B;
end Behavioral;
```

Rtl schematic:

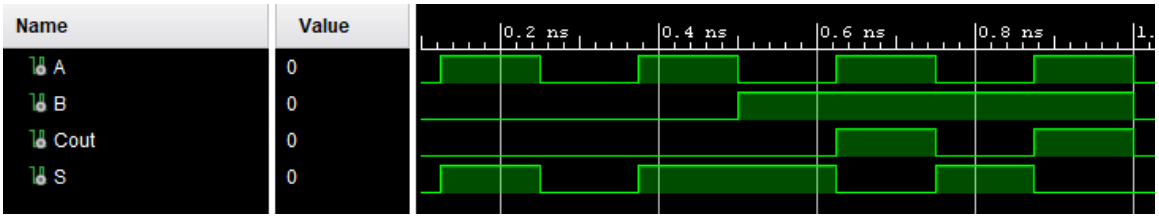


Testbench:

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

entity tb_halfadder is
end tb_halfadder;

architecture Behavioral of tb_halfadder is

component HalfAdder is
port(
A,B : in std_logic;
```

```
Cout,S: out std_logic);
end component;

signal A,B,Cout,S : std_logic :='0';

begin

ha: HalfAdder port map ( A => A, B =>B, Cout => Cout, S => S);

  proc: process(A,B) is
  begin
  A <= not A after 0.125ns;
  B <= not B after 0.5ns;
  end process proc;

end Behavioral;
```

Simulation:



‘Οπως βλέπουμε από την προσομοίωση, επαληθεύεται ο πίνακας αληθείας που παρατέθηκε πρωτύτερα.

Critical Path:

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	3	4	2	B	S	5.377	3.778	1.599
Path 2	∞	3	4	2	B	Cout	5.351	3.752	1.599

Άσκηση 2: Full Adder

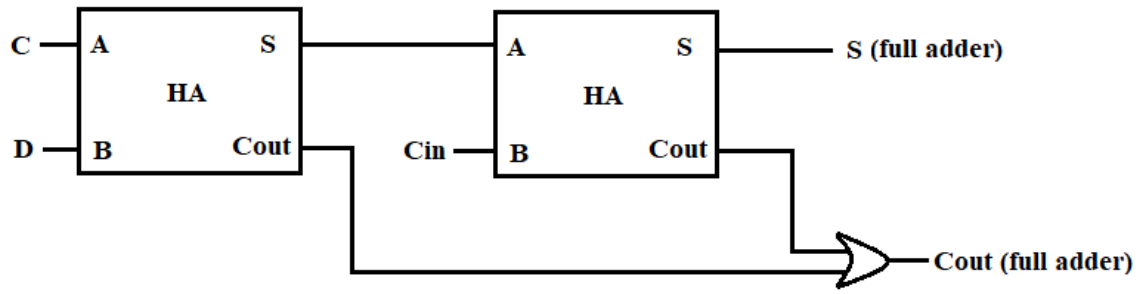
Cin	C	D	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Με βάση τον παραπάνω πίνακα αληθείας:

S=Cin’C’D+Cin’CD’+CinC’D’+CinCD=Cin’(C’D+CD’)+Cin(C’D’+CD)=Cin’(Xor(C,D))+Cin(xor(C,D))’=xor(Cin,xor(C,D))

Cout=Cin’CD+CinC’D+CinCD’+CinCD=Cin(xor(C,D))+CD

Με βάση αυτές τις δύο σχέσεις, και επειδή ο half adder αποτελείται από μία πύλη xor και μία πύλη and, συμπεραίνουμε πως μπορούμε να φτιάξουμε έναν full adder χρησιμοποιώντας δύο half adders και μία πύλη or, σύμφωνα με τον παρακάτω τρόπο:



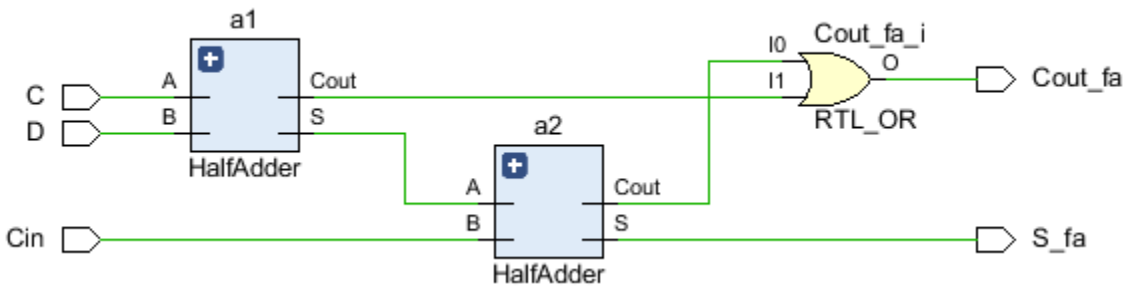
Κώδικας VHDL:

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

entity Full_adder is
port(
C,D,Cin : in std_logic;
S_fa, Cout_fa : out std_logic);
end Full_adder;

architecture Behavioral of Full_adder is

component HalfAdder is
port(
A,B : in std_logic;
Cout,S: out std_logic);
end component;

signal s1,s2,s3 : std_logic;

begin
```

```
a1: HalfAdder port map( A => C, B => D, S => s1, Cout => s2);

a2: HalfAdder port map( A => s1, B => Cin, S => S_fa, Cout => s3);


Cout_fa <= s3 or s2;


end Behavioral;
```

Έχωντας τον half adder έτοιμο από το προηγούμενο ζητούμενο, φτιάχνουμε τον full adder χρησιμοποιώντας τον half adder ως component και structural περιγραφή.

Rtl schematic:



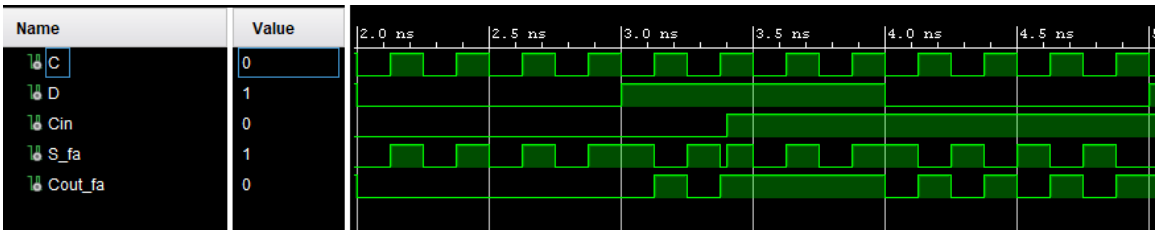
Βλέπουμε ότι το rtl schematic είναι σε συμφωνία με τις δύο σχέσεις που προέκυψαν από τον πίνακα αληθείας.

Testbench:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity tb_full_adder is

end tb_full_adder;
```

```
architecture Behavioral of tb_full_adder is

component Full_adder is
port(
C,D,Cin : in std_logic;
S_fa, Cout_fa : out std_logic);
end component;

signal C,D,Cin,S_fa,Cout_fa : std_logic := '0';

begin

fa : Full_adder port map ( C => C, D => D, Cin => Cin, S_fa => S_fa, Cout_fa => Cout_fa);

 proc: process(C,D,Cin) is
 begin
 C <= not C after 0.125ns;
 D <= not D after 1ns;
 Cin <= not Cin after 3.4ns;
 end process proc;
end Behavioral;
```

Simulation:



‘Οπως βλέπουμε από την προσομοίωση, επαληθεύεται ο πίνακας αληθείας που παρατέθηκε πρωτύτερα.

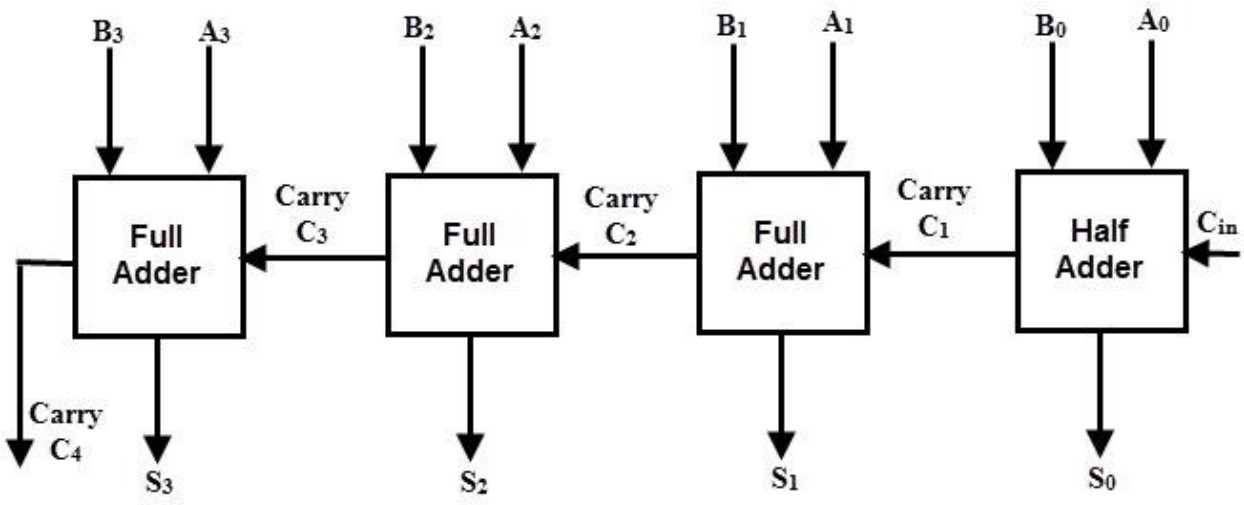
Critical Path:

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	3	4	2	D	Cout_fa	5.377	3.778	1.599
Path 2	∞	3	4	2	D	S_fa	5.351	3.752	1.599

Άσκηση 3:

Ζητουμενο της άσκησης είναι η υλοποίηση ενος αθροιστή των 4 bits (δηλαδή 2 αριθμών με εύρος τιμών απο 0 έως 15), κάνοντας χρήση του πλήρη αθροιστή του προηγούμενου ερωτήματος.

Παρατηρούμε ότι το κύκλωμα αυτό αποτελείται απο 4 τέτοιους αθροιστές συνδεδεμένους παράλληλα όπως φαίνεται στην παρακάτω εικόνα:

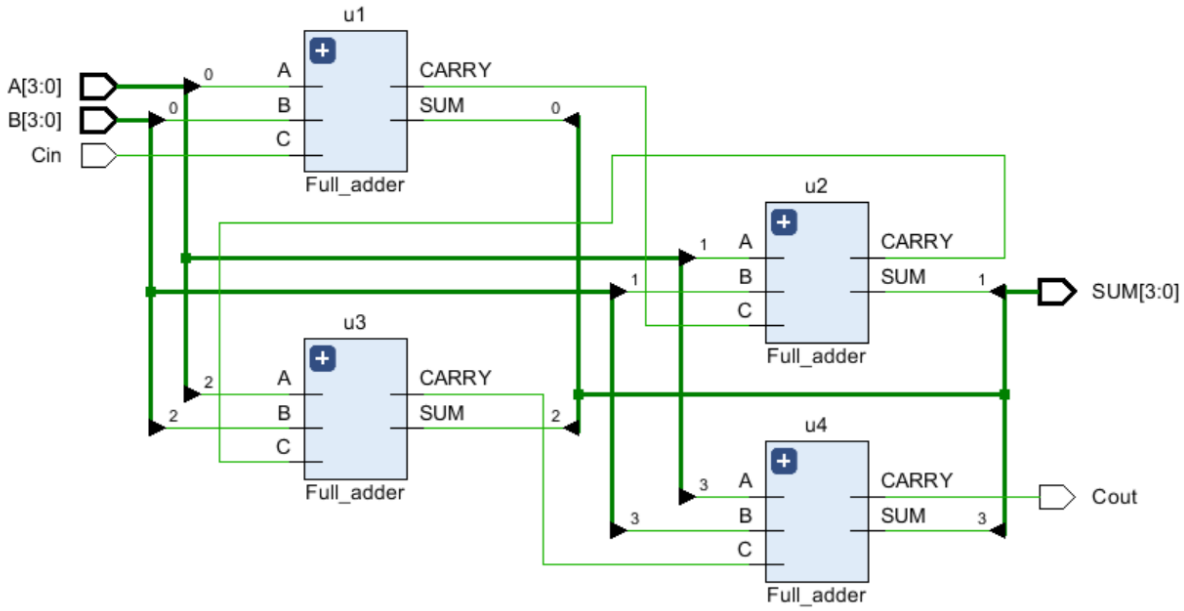


Ακολουθεί ο κώδικας που χρησιμοποιήθηκε για τη σχεδίαση του κυκλώματος :

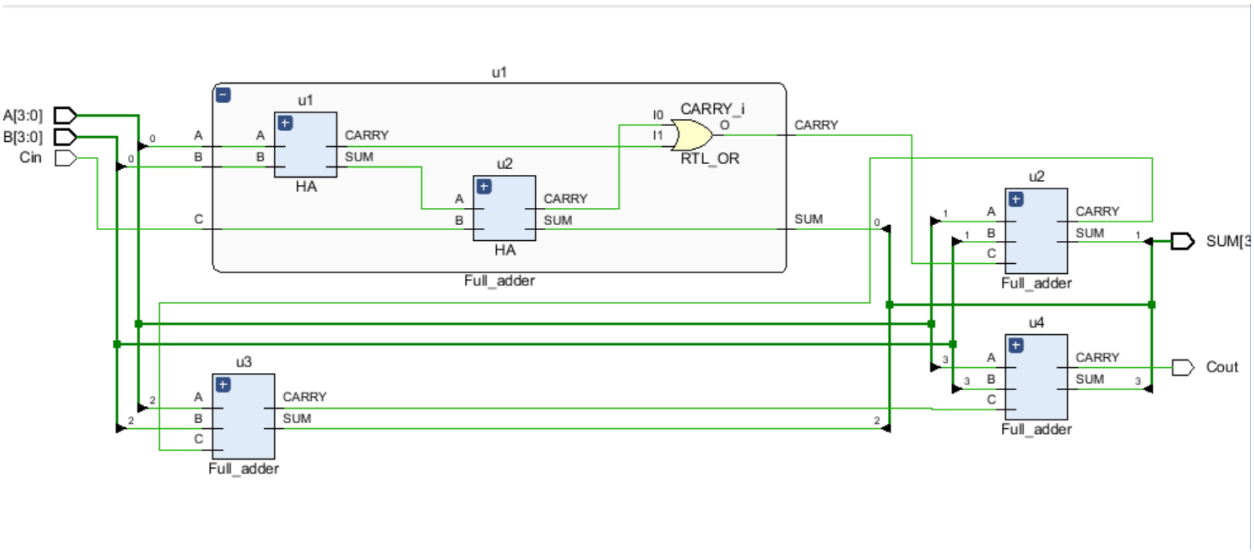
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity FourBitPA is
 PORT(
      A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cin : IN STD_LOGIC;
      SUM : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cout : OUT STD_LOGIC );
end FourBitPA;

architecture Structural of FourBitPA is

SIGNAL w0: STD_LOGIC;
SIGNAL w1: STD_LOGIC;
SIGNAL w2: STD_LOGIC;

COMPONENT fadd1
PORT(
      A : IN STD_LOGIC;
      B : IN STD_LOGIC;
      C : IN STD_LOGIC;
      SUM : OUT STD_LOGIC;
      CARRY : OUT STD_LOGIC );
END COMPONENT;

FOR ALL: fadd1 USE ENTITY work.Full_adder(Structural);

begin

u1 : fadd1 PORT MAP(A(0),B(0),Cin,SUM(0),w0);
u2 : fadd1 PORT MAP(A(1),B(1),w0,SUM(1),w1);
u3 : fadd1 PORT MAP(A(2),B(2),w1,SUM(2),w2);
u4 : fadd1 PORT MAP(A(3),B(3),w2,SUM(3),Cout);


end Structural;
```

Όπως αναφέραμε και παραπάνω στον κώδικα ορίζουμε ως component της structural περιγραφής,τον πλήρη αθροιστή του προηγούμενου ερωτήματος και στη συνέχεια δημιουργούμε 4 τέτοια αντίγραφα με το Cout του κάθε ενός να αποτελεί το Cin του επόμενου σταδίου.

Παραθέτουμε και το rtl διάγραμμα :



Παρατηρούμε ότι είναι ακριβώς όπως θα το περιμέναμε με 4 στάδια fadd όπου η κάθε μία είναι δομημένη στο εσωτερικό της όπως φαίνεται παρακάτω :

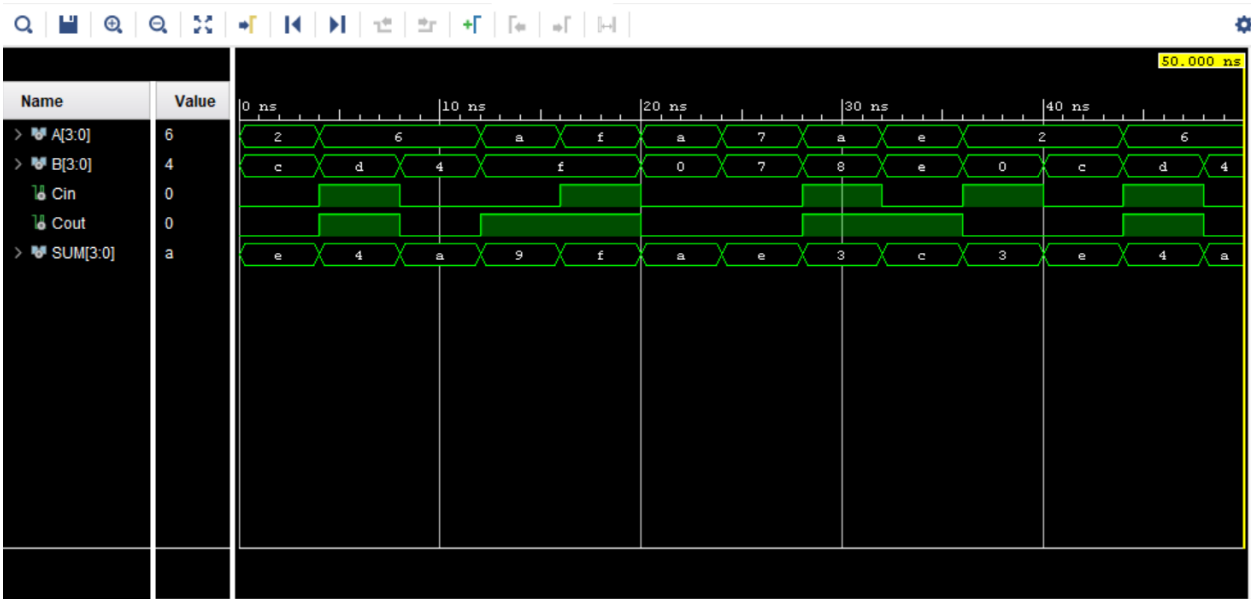


Το testebench που χρησιμοποιήθηκε για την επαλήθευση των αποτελεσμάτων δίνεται ως εξής :

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

entity FourBitPA_tb is
end FourBitPA_tb;

architecture Structural of FourBitPA_tb is


SIGNAL   A: std_logic_vector(3 downto 0) := "0000";
SIGNAL   B: std_logic_vector(3 downto 0) := "0000";
SIGNAL   Cin: std_logic := '0';
SIGNAL   Cout: std_logic;
SIGNAL   SUM: std_logic_vector(3 downto 0);

begin

UUT : entity work.FourBitPA port map(A,B,Cin,SUM,Cout);

tb: PROCESS
begin
```

```
A <= "0010";
B <= "1100";
Cin <= '0';
wait for 4ns;
A <= "0110";
B <= "1101";
Cin <= '1';
wait for 4ns;
A <= "0110";
B <= "0100";
Cin <= '0';
wait for 4ns;
A <= "1010";
B <= "1111";
Cin <= '0';
wait for 4ns;
A <= "1111";
B <= "1111";
Cin <= '1';
wait for 4ns;
A <= "1010";
B <= "0000";
Cin <= '0';
wait for 4ns;
A <= "0111";
B <= "0111";
Cin <= '0';
wait for 4ns;
A <= "1010";
B <= "1000";
Cin <= '1';
wait for 4ns;
A <= "1110";
B <= "1110";
Cin <= '0';
wait for 4ns;
A <= "0010";
B <= "0000";
Cin <= '1';
wait for 4ns;

end process;

end Structural;
```

Και ακολουθούν οι κυματομορφές που παράγονται μέσω της προσομοίωσης :



Η τιμές είναι οι θεωρητικά αναμενόμενες,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	∞	4	5	3	A1	Cout	5.942	3.876	2.066
Path 2	∞	4	5	3	A1	SUM[2]	5.942	3.876	2.066
Path 3	∞	4	5	3	A1	SUM[3]	5.936	3.870	2.066
Path 4	∞	3	4	3	B0	SUM[1]	5.379	3.780	1.599
Path 5	∞	3	4	3	B0	SUM[0]	5.351	3.752	1.599

όπου οι τιμές των καθυστερήσεων είναι εντός των αναμενόμενων ορίων.

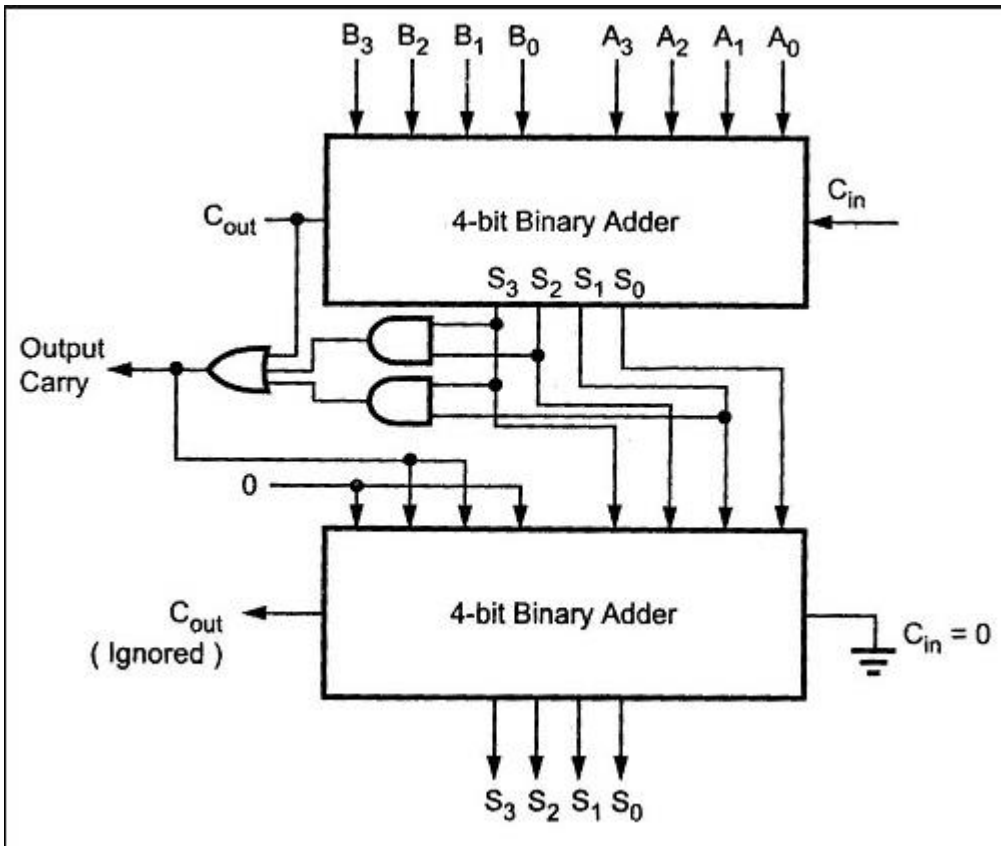
Άσκηση 4:

Ζητούμενο της άσκησης είναι η υλοποίηση ενός BCD αθροιστή.Η λειτρουγία ενός τέτοιου αθροιστή είναι η πρόσθεση δύο αριθμών με εύρος τιμών από 0 έως 9 και η εμφάνιση του αποτελέσματος ως εξής :

cout = 1 εάν το άθροισμα υπερβεί τον αριθμό 9, διαφορετικά 0

sum = το άθροισμα των αριθμών αν είναι μικρότερο του 10, διαφορετικά τον αριθμό : άθροισμα - 10

Το σχηματικό διάγραμμα ενός τέτοιου αθροιστή δίνεται παρακάτω :



Παρατηρούμε ότι ο αθροιστής αυτός αποτελείται από 2 μονάδες 4-bit Parallel Adder της προηγούμενης άσκησης σε συνδυασμό με 2 πύλες AND και μία πύλη OR για το Cout. Επιπλέον,παρατηρούμε ανάδραση στο σήμα του Cout επομένως θα το ορίσουμε ως INOUT.

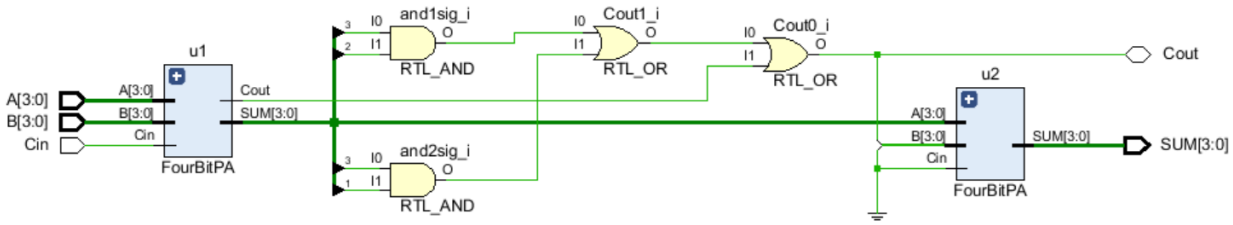
Ο κώδικας περγραφής του κυκλώματος:

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


entity BCDadder is
PORT(
      A : IN STD_LOGIC_VECTOR(3 DOWNTO 0 );
      B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cin : IN STD_LOGIC;
      SUM : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cout : INOUT STD_LOGIC );
end BCDadder;

architecture Structural of BCDadder is

SIGNAL sumsig: STD_LOGIC_Vector(3 downto 0);
SIGNAL carrysig: STD_LOGIC;
SIGNAL and1sig: STD_LOGIC;
SIGNAL and2sig: STD_LOGIC;
SIGNAL zero: STD_LOGIC := '0';
SIGNAL irrelevant: STD_LOGIC;

COMPONENT FourBitPA is
 PORT(
      A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cin : IN STD_LOGIC;
      SUM : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cout : OUT STD_LOGIC );
end COMPONENT;

FOR ALL: FourBitPA USE ENTITY work.FourBitPA(Structural);

begin

u1 : FourBitPA PORT MAP(A,B,Cin,sumsig,carrysig);
and1sig <= sumsig(3) AND sumsig(2);
and2sig <= sumsig(3) AND sumsig(1);
Cout <= and1sig OR and2sig OR carrysig;
u2 : FourBitPA PORT MAP(A => sumsig,
                        B(0) => zero,
                        B(1) =>Cout,
                        B(2) =>Cout,
                        B(3)=> zero,
                        Cin => zero,
```

```
                    SUM => SUM,
                    Cout => irrelevant);

end Structural;
```

Και πάλι βλέπουμε πως για την υλοποίηση του θέλουμε ως component τον 4-bitPA, χρησιμοποιώντας το 2 φορές με τις αντίστοιχες πύλες να παράγουν το Cout από τα αποτελέσματα του πρώτου 4-bitPA.

Το rtl διάγραμμα που παράγεται είναι όπως ακριβώς δόθηκε παραπάνω στο σχηματικό διάγραμμα και φαίνεται παρακάτω :

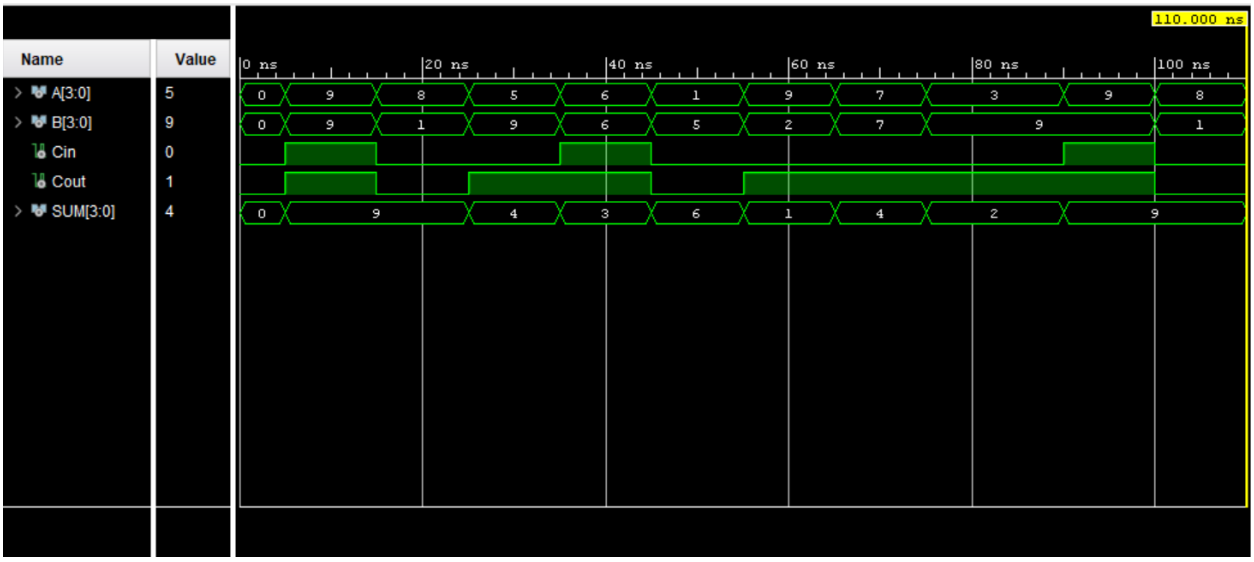


Το testebench που χρησιμοποιήθηκε για την επαλήθευση των αποτελεσμάτων δίνεται ως εξής :

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

entity BCD_tb is
end BCD_tb;

architecture Structural of BCD_tb is

SIGNAL   A: std_logic_vector(3 DOWNTO 0) := "0000";
SIGNAL   B: std_logic_vector(3 DOWNTO 0) := "0000";
SIGNAL   Cin: std_logic := '0';
SIGNAL   Cout: std_logic;
```

```
SIGNAL   SUM: std_logic_vector(3 downto 0);

begin

UUT : entity work.BCDadder port map(A,B,Cin,SUM,Cout);

tb: PROCESS
begin

wait for 5 ns;
A <= "1001";
B <= "1001";
Cin <= '1';
wait for 10 ns;
A <= "1000";
B <= "0001";
Cin <= '0';
wait for 10 ns;
A <= "0101";
B <= "1001";
Cin <= '0';
wait for 10 ns;
A <= "0110";
B <= "0110";
Cin <= '1';
wait for 10 ns;
A <= "0001";
B <= "0101";
Cin <= '0';
wait for 10 ns;
A <= "1001";
B <= "0010";
Cin <= '0';
wait for 10 ns;
A <= "0111";
B <= "0111";
Cin <= '0';
wait for 10 ns;
A <= "0011";
B <= "1001";
Cin <= '0';
wait for 10 ns;

end process;


end Structural;
```

Και ακολουθούν ο κυματομορφές που παράγονται μέσω της προσομοίωσης :



Οι τιμές είναι οι θεωρητικά αναμενόμενες,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	∞	7	8	5	A[1]	SUM[3]	7.695	4.248	3.447
Path 2	∞	6	7	5	A[1]	SUM[1]	7.122	4.124	2.998
Path 3	∞	6	7	5	A[1]	SUM[2]	7.116	4.118	2.998
Path 4	∞	5	6	5	A[1]	Cout	6.525	4.000	2.525
Path 5	∞	3	4	3	B[0]	SUM[0]	5.351	3.752	1.599

Και πάλι,οι τιμές των καθυστερήσεων είναι εντός των αναμενόμενων ορίων.

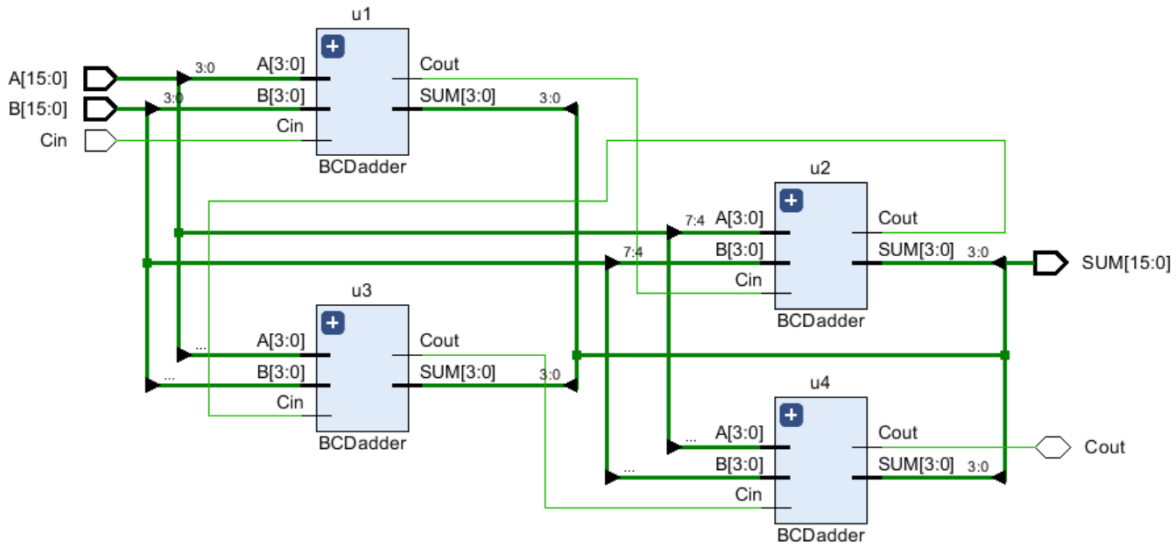
Άσκηση 5:

Ζητούμενο της άσκησης είναι η επέκταση του προηγούμενου ζητήματος έτσι ώστε να κατασκευαστεί ένας BCD αθροιστής των 4 ψηφίων (δηλαδή να προσθέτει δύο αριθμούς με εύρος τιμών για τον καθένα από 0000 έως 9999. Η υλοποίηση είναι εντελώς αντίστοιχη με αυτή της άσκησης 3,δηλαδή και σε αυτήν την περίπτωση αρκεί απλά να προσθέσουμε εν παραλλήλω 4 BCD αθροιστές με το κρατούμενο εξόδου του κάθε ενός να αποτελεί το κρατούμενο εισόδου του επόμενου.Ο κώδικας που χρησμοποιήθηκε δίνεται παρακάτω:

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

entity FourDigitBCD is
PORT(
      A : IN STD_LOGIC_VECTOR(15 DOWNTO 0 );
      B : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      Cin : IN STD_LOGIC;
      SUM : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
      Cout : INOUT STD_LOGIC );
end FourDigitBCD;

architecture Structural of FourDigitBCD is

SIGNAL carrysig: STD_LOGIC_VECTOR(2 DOWNTO 0);

COMPONENT BCDadder is
PORT(
      A : IN STD_LOGIC_VECTOR(3 DOWNTO 0 );
      B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cin : IN STD_LOGIC;
      SUM : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      Cout : INOUT STD_LOGIC );
end COMPONENT;

FOR ALL: BCDadder USE ENTITY work.BCDadder(Structural);

begin

u1 : BCDadder PORT MAP(A(3 DOWNTO 0),B(3 DOWNTO 0),Cin,SUM(3
DOWNTO 0),carrysig(0));
```

```
u2 : BCDadder PORT MAP(A(7 DOWNTO 4),B(7 DOWNTO
4),carrysig(0),SUM(7 DOWNTO 4),carrysig(1));
u3 : BCDadder PORT MAP(A(11 DOWNTO 8),B(11 DOWNTO
8),carrysig(1),SUM(11 DOWNTO 8),carrysig(2));
u4 : BCDadder PORT MAP(A(15 DOWNTO 12),B(15 DOWNTO
12),carrysig(2),SUM(15 DOWNTO 12),Cout);


end Structural;
```

όπου γίνεται η χρήση του component BCD-adder 4 φορές.Το rtl διάγραμμα που προκύπτει είναι το εξής :



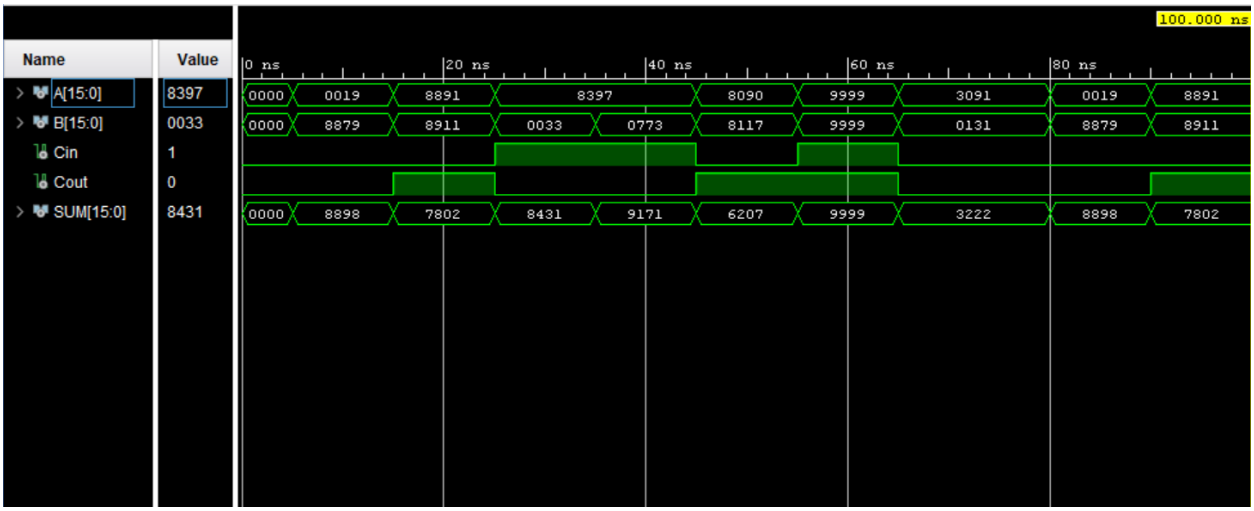
και είναι ακριβώς αυτό που περιμέναμε,4 BCD-adders παράλληλα.

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

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


entity FourDigitBCD_tb is
end FourDigitBCD_tb;

architecture Structural of FourDigitBCD_tb is

SIGNAL   A: std_logic_vector(15 DOWNTO 0) := "0000000000000000";
SIGNAL   B: std_logic_vector(15 DOWNTO 0) := "0000000000000000";
SIGNAL   Cin: std_logic := '0';
SIGNAL   Cout: std_logic;
SIGNAL   SUM: std_logic_vector(15 downto 0);


begin

UUT : entity work.FourDigitBCD port map(A,B,Cin,SUM,Cout);

tb: PROCESS
begin

wait for 5 ns;
A <= "0000000000011001";
B <= "1000100001111001";
Cin <= '0';
wait for 10ns;
A <= "1000100010010001";
B <= "1000100100010001";
Cin <= '0';
wait for 10ns;
A <= "1000001110010111";
B <= "0000000000110011";
Cin <= '1';
wait for 10ns;
A <= "1000001110010111";
B <= "0000011101110011";
Cin <= '1';
wait for 10ns;
A <= "1000000010010000";
B <= "1000000100010111";
Cin <= '0';
wait for 10ns;
```

```
A <= "1001100110011001";
B <= "1001100110011001";
Cin <= '1';
wait for 10ns;
A <= "0011000010010001";
B <= "0000000100110001";
Cin <= '0';
wait for 10ns;

end process;

end Structural;
```

Και ακολουθούν ο κυματομορφές που παράγονται μέσω της προσομοίωσης :



Η τιμές είναι οι θεωρητικά αναμενόμενες,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	∞	13	14	7	B[0]	SUM[14]	11.268	5.002	6.266
Path 2	∞	13	14	7	B[0]	SUM[15]	11.268	5.002	6.266
Path 3	∞	12	13	7	B[0]	SUM[13]	10.695	4.878	5.817
Path 4	∞	11	12	7	B[0]	Cout	10.098	4.754	5.344
Path 5	∞	9	10	7	B[0]	SUM[10]	8.947	4.512	4.435
Path 6	∞	9	10	7	B[0]	SUM[9]	8.947	4.512	4.435
Path 7	∞	9	10	7	B[0]	SUM[12]	8.941	4.506	4.435
Path 8	∞	8	9	7	B[0]	SUM[11]	8.345	4.394	3.951
Path 9	∞	7	8	7	B[0]	SUM[5]	7.761	4.270	3.491
Path 10	∞	7	8	7	B[0]	SUM[6]	7.761	4.270	3.491

Οι τιμές των καθυστερήσεων είναι αναμενόμενα μεγαλύτερες από αυτές των προηγούμενων κυκλωμάτων,όντας ωστόσο σε λογικές τιμές.