Ψηφιακά Συστήματα **VLSI**

8ο εξάμηνο, Ακαδημαϊκή περίοδος 2024-2025

2η Εργαστηριακή Άσκηση

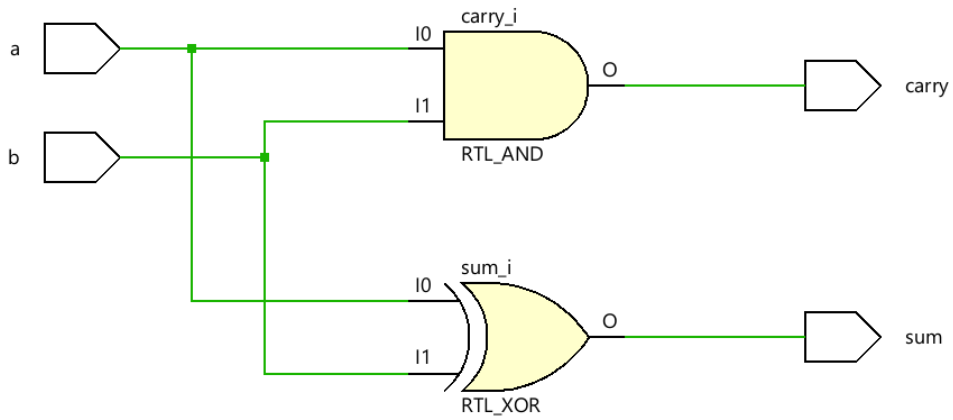
Δημήτρης Καμπανάκης: 03121012
Αγγελική Ζέρβα: 03121101

Θέμα 1: Ημιαθροιστής σε περιγραφή dataflow

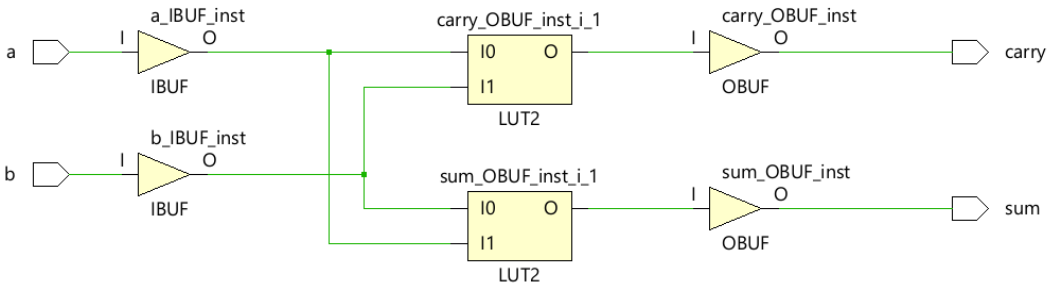
Ο κώδικας σε vhdl για την περιγραφή του ημιαθροιστή και της αρχιτεκτονικής του σε dataflow vhdl είναι ο εξής:

```
library ieee;

use ieee.std_logic_1164.all;

entity half_adder is

    port(
        a : in std_logic;
        b : in std_logic;
        sum : out std_logic;
        carry : out std_logic
    );
end half_adder;

-- dataflow
architecture half_adder_arch of half_adder is
begin
    sum <= a xor b;
    carry <= a and b;

end half_adder_arch;
```

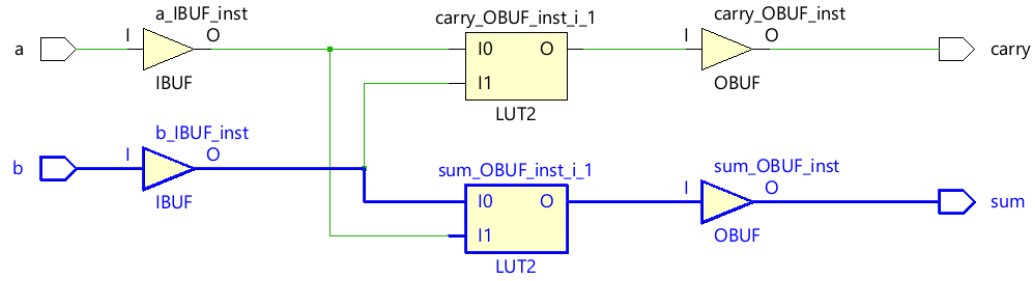
Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis παίρνουμε το εξής σχηματικό:



Χρησιμοποιώντας το timing summary για το κύκλωμα μας μπορούμε να βρούμε το κρίσιμο μονοπάτι του κυκλώματος μας, το οποίο φαίνεται παρακάτω:



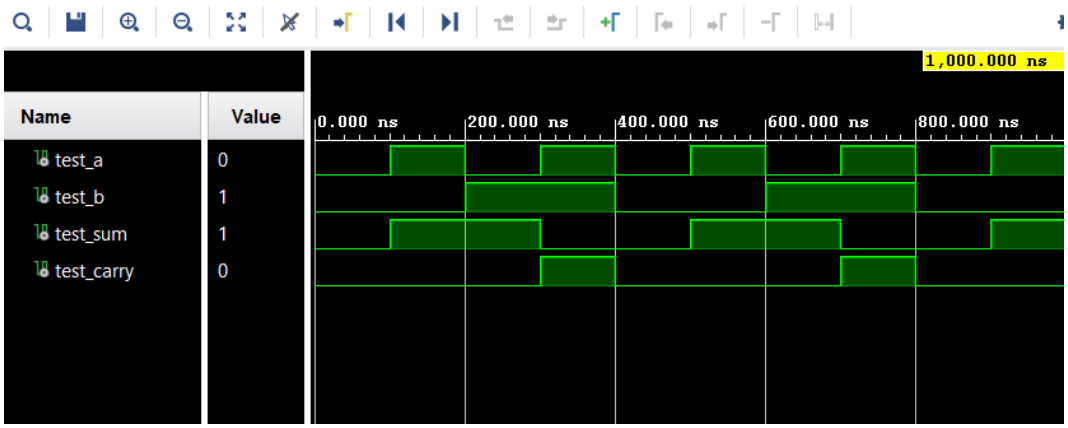
Όπως φαίνεται παρακάτω η χρονική καθυστέρηση του critical path του κυκλώματος είναι 5.377 ns.

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	3	4	2	b	sum	5.377	3.778	1.599	∞	input port clock		
Path 2	∞	3	4	2	b	sum	5.377	3.778	1.599	∞	input port clock		
Path 3	∞	3	4	2	a	carry	5.351	3.752	1.599	∞	input port clock		
Path 4	∞	3	4	2	b	carry	5.351	3.752	1.599	∞	input port clock		

Ο εντοπισμός του κρίσιμου μονοπατιού είναι σημαντικός, καθώς μπορούμε να παρατηρήσουμε ποιο μονοπάτι δεδομένων είναι το πιο χρονοβόρο στην σχεδίαση μας. Έτσι, αν χρειαστούμε βελτιώσεις στον χρόνο ξέρουμε ποιο μονοπάτι χρειάζεται να αλλάξει. Επιπλέον, η μέγιστη χρονική καθυστέρηση επηρεάζει άμεσα την επιλογή της ελάχιστης συχνότητας ρολογιού.
Για την προσομοίωση και τον έλεγχο της σωστής λειτουργίας του κυκλώματός μας χρησιμοποιούμε το παρακάτω testbench:

```
library ieee;

use ieee.std_logic_1164.all;

entity half_adder_tb is
end half_adder_tb;

architecture half_adder_tb_arch of half_adder_tb is

component half_adder
        port(
        a : in std_logic;
        b : in std_logic;
        sum : out std_logic;
```

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

-- stimulus signals

signal test_a   : std_logic;
signal test_b : std_logic;
signal test_sum : std_logic;
signal test_carry : std_logic;

begin

uut: half_adder
    port map(
        a => test_a,
        b => test_b,
        sum => test_sum,
        carry => test_carry
    );

 testing : process

 begin

    test_a <= '0';
    test_b <= '0';
    wait for 100 ns;

    test_a <= '1';
    test_b <= '0';
    wait for 100 ns;

    test_a <= '0';
    test_b <= '1';
    wait for 100 ns;

    test_a <= '1';
    test_b <= '1';
    wait for 100 ns;

end process;
end half_adder_tb_arch;
```



Παρατηρούμε ότι έχουμε ορθά αποτελέσματα για όλους τους συνδυασμούς εισόδων.

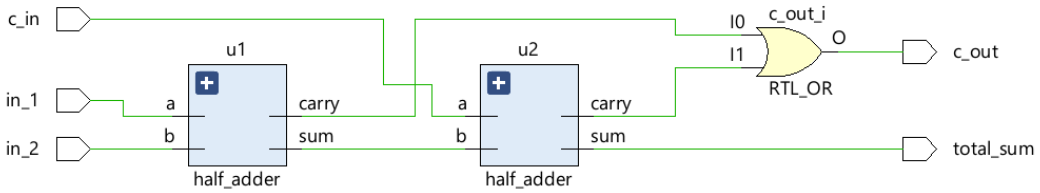
Θέμα 2: Πλήρης αθροιστής σε περιγραφή structural

Ο κώδικας σε vhdl για την περιγραφή του αθροιστή και της αρχιτεκτονικής του σε structural vhdl είναι ο εξής:

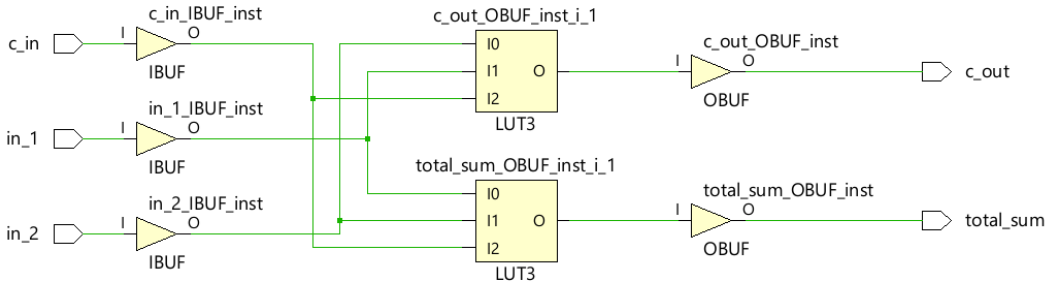
```
library ieee;

use ieee.std_logic_1164.all;

entity full_adder is
    port(
        in_1 : in std_logic;
        in_2 : in std_logic;
        c_in : in std_logic;
        total_sum : out std_logic;
        c_out : out std_logic
    );
end entity;

-- structural architecture
architecture full_adder_arch of full_adder is

    signal sub_sum : std_logic;
    signal c_out_1 : std_logic;
    signal c_out_2 : std_logic;

    component half_adder is
        port(
        a : in std_logic;
        b : in std_logic;
        sum : out std_logic;
        carry : out std_logic
        );
     end component;

     begin

        u1: half_adder port map
```

```
        (a => in_1,
        b => in_2,
        sum => sub_sum,
        carry => c_out_1
    );

    u2: half_adder port map
        (a => c_in,
        b => sub_sum,
        sum => total_sum,
        carry => c_out_2
    );

    c_out <= c_out_1 or c_out_2;

end full_adder_arch;
```

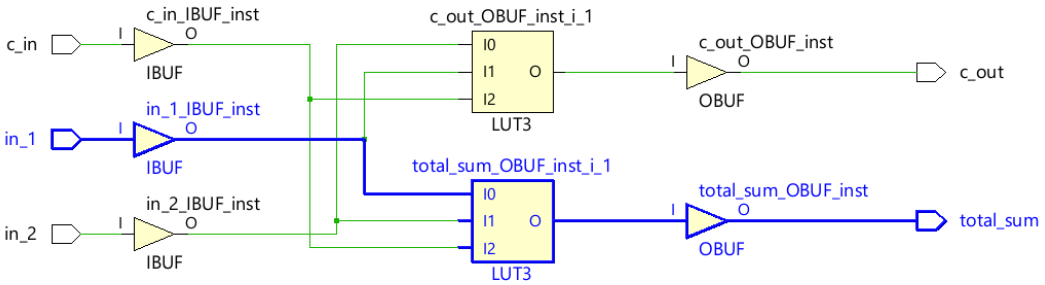
Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Μπορούμε να παρατηρήσουμε ότι όπως αναμένεται στο elaborated design ο πλήρης αθροιστής αποτελείται από 2 ημιαθροιστές και μία πύλη or.
Με τη χρήση του synthesis παίρνουμε το εξής σχηματικό:



Χρησιμοποιώντας το timing summary για το κύκλωμα μας μπορούμε να βρούμε το κρίσιμο μονοπάτι του κυκλώματος μας, το οποίο φαίνεται παρακάτω:



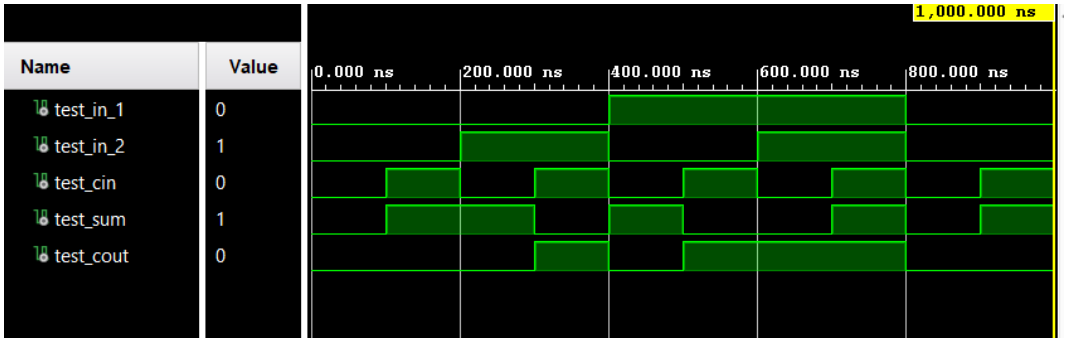
Όπως φαίνεται παρακάτω η χρονική καθυστέρηση του critical path του κυκλώματος είναι 5.377 ns.

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Excep
Path 1	∞	3	4	2	in_1	total_sum	5.377	3.778	1.599	∞	input port clock		
Path 2	∞	3	4	2	in_1	total_sum	5.377	3.778	1.599	∞	input port clock		
Path 3	∞	3	4	2	in_2	total_sum	5.377	3.778	1.599	∞	input port clock		
Path 4	∞	3	4	2	in_2	c_out	5.351	3.752	1.599	∞	input port clock		
Path 5	∞	3	4	2	in_1	c_out	5.351	3.752	1.599	∞	input port clock		
Path 6	∞	3	4	2	c_in	c_out	5.351	3.752	1.599	∞	input port clock		

Για την προσομοίωση και τον έλεγχο της σωστής λειτουργίας του κυκλώματός μας χρησιμοποιούμε το παρακάτω testbench:

```
library ieee;

use ieee.std_logic_1164.all;

entity full_adder_tb is
end full_adder_tb;

architecture full_adder_tb_arch of full_adder_tb is

component full_adder
        port(
        in_1 : in std_logic;
        in_2 : in std_logic;
        c_in : in std_logic;
        total_sum : out std_logic;
        c_out : out std_logic
    );
end component;

-- stimulus signals

signal test_in_1   : std_logic;
signal test_in_2 : std_logic;
signal test_cin : std_logic;
signal test_sum : std_logic;
signal test_cout : std_logic;

begin

uut: full_adder
    port map(
```

```
        in_1 => test_in_1,
        in_2 => test_in_2,
        total_sum => test_sum,
        c_in => test_cin,
        c_out => test_cout
    );

 testing : process

 begin

    test_in_1 <= '0';
    test_in_2 <= '0';
    test_cin <= '0';
    wait for 100 ns;

    test_in_1 <= '0';
    test_in_2 <= '0';
    test_cin <= '1';
    wait for 100 ns;

    test_in_1 <= '0';
    test_in_2 <= '1';
    test_cin <= '0';
    wait for 100 ns;

    test_in_1 <= '0';
    test_in_2 <= '1';
    test_cin <= '1';
    wait for 100 ns;

    test_in_1 <= '1';
    test_in_2 <= '0';
    test_cin <= '0';
    wait for 100 ns;

    test_in_1 <= '1';
    test_in_2 <= '0';
    test_cin <= '1';
    wait for 100 ns;

    test_in_1 <= '1';
    test_in_2 <= '1';
    test_cin <= '0';
    wait for 100 ns;

    test_in_1 <= '1';
    test_in_2 <= '1';
    test_cin <= '1';
    wait for 100 ns;

end process;
end full_adder_tb_arch;
```



Παρατηρούμε ότι έχουμε ορθά αποτελέσματα για όλους τους συνδυασμούς εισόδων.

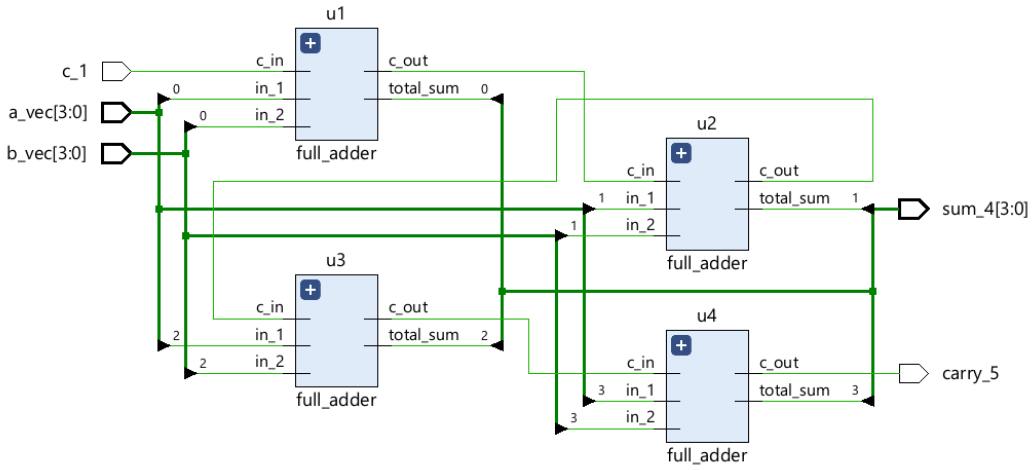
Θέμα 3: Παράλληλος αθροιστής των 4 bit σε περιγραφή structural

Ο κώδικας σε vhdl για την περιγραφή του παράλληλου αθροιστή και της αρχιτεκτονικής του σε structural vhdl είναι ο εξής:

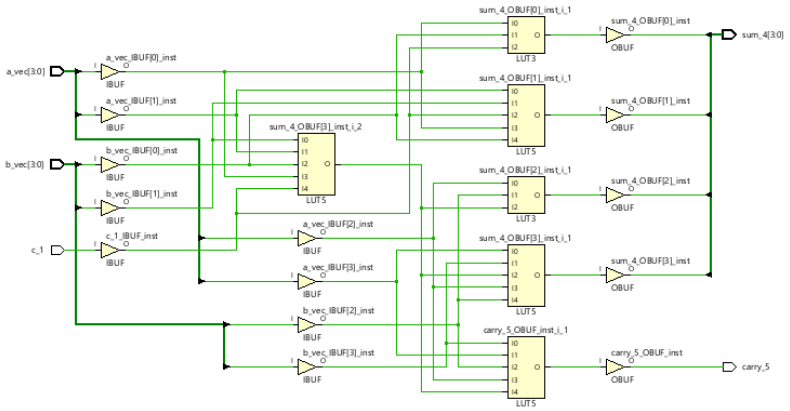
```
library ieee;

use ieee.std_logic_1164.all;

entity parallel_4bit_adder is
    port(
        a_vec : in std_logic_vector(4-1 downto 0);
        b_vec : in std_logic_vector(4-1 downto 0);
        c_1 : in std_logic;
        sum_4 : out std_logic_vector(4-1 downto 0);
        carry_5 : out std_logic
    );
end entity;

-- structural architecture
architecture parallel_4bit_adder_arch of parallel_4bit_adder is

    signal c_2 : std_logic;
    signal c_3 : std_logic;
    signal c_4 : std_logic;


    component full_adder is
    port(
        in_1 : in std_logic;
        in_2 : in std_logic;
        c_in : in std_logic;
        total_sum : out std_logic;
        c_out : out std_logic
    );
```

```
    end component;

    begin

       u1: full_adder port map
           (in_1 => a_vec(0),
           in_2 => b_vec(0),
           c_in => c_1,
           total_sum => sum_4(0),
           c_out => c_2
        );

       u2: full_adder port map
           (in_1 => a_vec(1),
           in_2 => b_vec(1),
           c_in => c_2,
           total_sum => sum_4(1),
           c_out => c_3
        );

         u3: full_adder port map
           (in_1 => a_vec(2),
           in_2 => b_vec(2),
           c_in => c_3,
           total_sum => sum_4(2),
           c_out => c_4
        );

         u4: full_adder port map
           (in_1 => a_vec(3),
           in_2 => b_vec(3),
           c_in => c_4,
           total_sum => sum_4(3),
           c_out => carry_5
        );
end parallel_4bit_adder_arch;
```

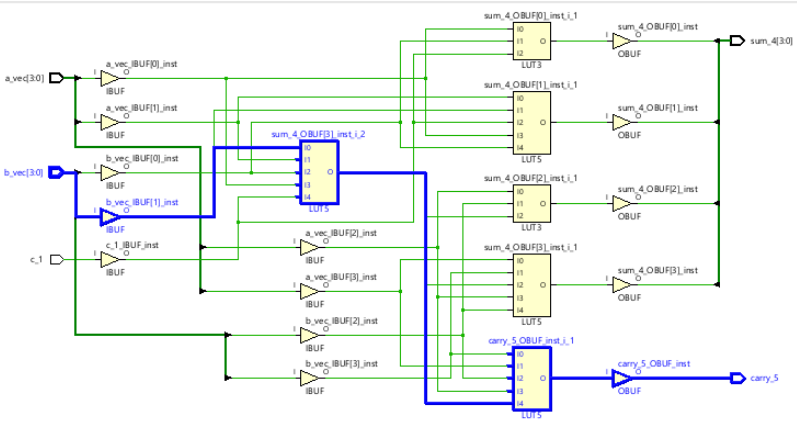
Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Μπορούμε να παρατηρήσουμε ότι όπως αναμένεται στο elaborated design ο παράλληλος αθροιστής αποτελείται από 4 πλήρεις αθροιστές.
Με τη χρήση του synthesis παίρνουμε το εξής σχηματικό:



Χρησιμοποιώντας το timing summary για το κύκλωμα μας μπορούμε να βρούμε το κρίσιμο μονοπάτι του κυκλώματος μας, το οποίο φαίνεται παρακάτω:



Όπως φαίνεται παρακάτω η χρονική καθυστέρηση του critical path του κυκλώματος είναι 5.970 ns.

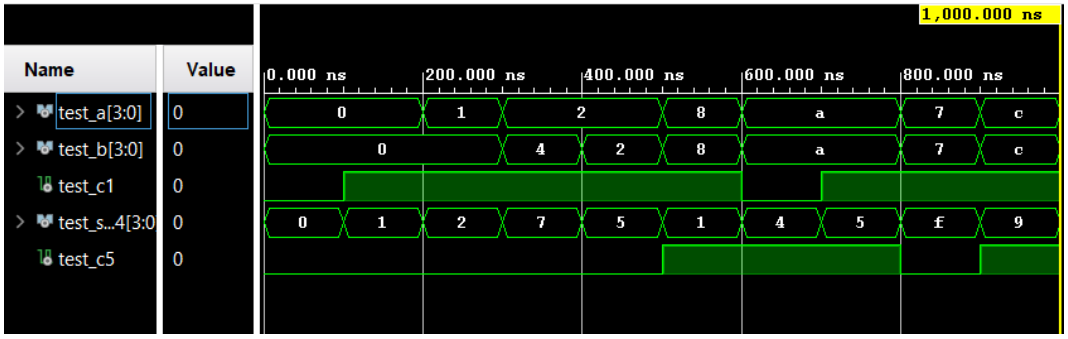
Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	∞	4	5	3	b_vec[1]	carry_5	5.970	3.904	2.066	∞	input port clock	
Path 2	∞	4	5	3	a_vec[1]	carry_5	5.970	3.904	2.066	∞	input port clock	
Path 3	∞	4	5	3	b_vec[1]	carry_5	5.970	3.904	2.066	∞	input port clock	
Path 4	∞	4	5	3	a_vec[1]	carry_5	5.970	3.904	2.066	∞	input port clock	
Path 5	∞	4	5	3	b_vec[1]	sum_4[2]	5.970	3.904	2.066	∞	input port clock	
Path 6	∞	4	5	3	a_vec[1]	sum_4[2]	5.970	3.904	2.066	∞	input port clock	
Path 7	∞	4	5	3	b_vec[1]	sum_4[2]	5.970	3.904	2.066	∞	input port clock	

Για την προσομοίωση και τον έλεγχο της σωστής λειτουργίας του κυκλώματός μας χρησιμοποιούμε το παρακάτω testbench:

```
library ieee;

use ieee.std_logic_1164.all;

entity parallel_4bit_adder_tb is
end parallel_4bit_adder_tb;
```

```
architecture parallel_4bit_adder_tb_arch of parallel_4bit_adder_tb is

component parallel_4bit_adder
    port(
        a_vec : in std_logic_vector(4-1 downto 0);
        b_vec : in std_logic_vector(4-1 downto 0);
        c_1 : in std_logic;
        sum_4 : out std_logic_vector(4-1 downto 0);
        carry_5 : out std_logic
    );
end component;

-- stimulus signals

signal test_a   : std_logic_vector(4-1 downto 0);
signal test_b : std_logic_vector(4-1 downto 0);
signal test_c1 : std_logic;
signal test_sum4 : std_logic_vector(4-1 downto 0);
signal test_c5 : std_logic;

begin

uut: parallel_4bit_adder
    port map(
        a_vec => test_a,
        b_vec => test_b,
        sum_4 => test_sum4,
        c_1 => test_c1,
        carry_5 => test_c5
    );

 testing : process

 begin

    test_a <= "0000";
    test_b <= "0000";
    test_c1 <= '0';
    wait for 100 ns;

    test_a <= "0000";
    test_b <= "0000";
    test_c1 <= '1';
    wait for 100 ns;

    test_a <= "0001";
    test_b <= "0000";
    test_c1 <= '1';
    wait for 100 ns;

    test_a <= "0010";
    test_b <= "0100";
    test_c1 <= '1';
```

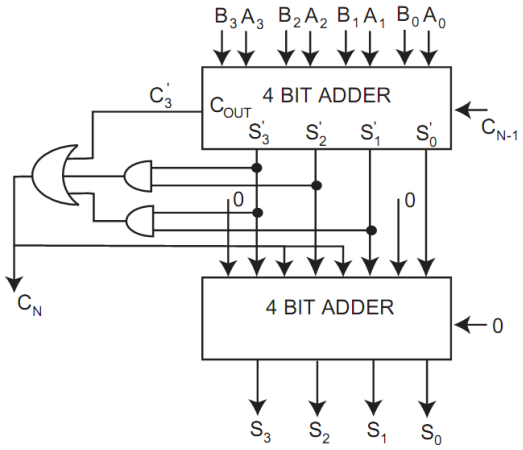
```
    wait for 100 ns;


    test_a <= "0010";
    test_b <= "0010";
    test_c1 <= '1';
    wait for 100 ns;


    test_a <= "1000";
    test_b <= "1000";
    test_c1 <= '1';
    wait for 100 ns;

    test_a <= "1010";
    test_b <= "1010";
    test_c1 <= '0';
    wait for 100 ns;


    test_a <= "1010";
    test_b <= "1010";
    test_c1 <= '1';
    wait for 100 ns;


    test_a <= "0111";
    test_b <= "0111";
    test_c1 <= '1';
    wait for 100 ns;

    test_a <= "1100";
    test_b <= "1100";
    test_c1 <= '1';
    wait for 100 ns;




end process;
end parallel_4bit_adder_tb_arch;
```



Παρατηρούμε ότι έχουμε ορθά αποτελέσματα για όλους τους συνδυασμούς εισόδων που δίνονται. Σημειώνεται ότι δεν είναι όλοοι οι πιθανοί συνδυασμοί εισόδων, αλλά είναι οι συνδυασμοί που πιθανόν να παρουσιαστούν σφάλματα, οπότε είναι αντιπροσωπευτικοί για την ορθότητα του κυκλώματος.

Θέμα 3: BCD Full Adder σε περιγραφή structural

Για τη σχεδίαση του bcd αθροιστή βασιζόμαστε στο εξής σχηματικό:



Στην BCD αναπαράσταση κάθε ψηφίο του αριθμού αναπαρίσταται σε binary μορφή. Για παράδειγμα, ο αριθμός 12 σε BCD θα είναι 0001 0010. Για τη λειτουργία του αθροιστή ισχύει ότι όταν το αποτέλεσμα της πρόσθεσης είναι μικρότερο ή ίσο του 9 τότε το cout είναι 0, ενώ αν το αποτέλεσμα της πρόσθεσης είναι μεγαλύτερο του 9 τότε το cout του πρώτου adder είναι 1. Τότε, όπως φαίνεται και στο rtl design προστίθεται στο άθροισμα του αριθμού που προκύπτει από τον πρώτο adder ο αριθμός 6. Έτσι προκύπτει το σωστό αποτέλεσμα.

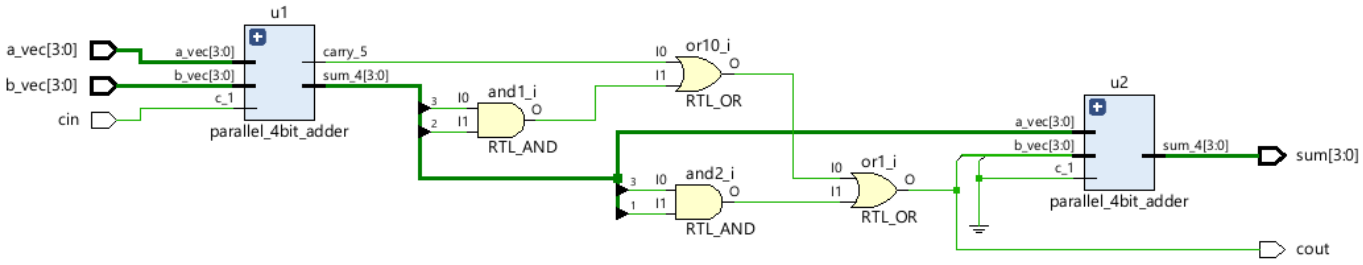
Ο κώδικας σε vhdl για την περιγραφή του bcd full adder και της αρχιτεκτονικής του σε structural vhdl είναι ο εξής:

```
library ieee;

use ieee.std_logic_1164.all;

entity bcd_full_adder is
    Port(
        a_vec : in std_logic_vector (3 downto 0);
        b_vec : in std_logic_vector (3 downto 0);
        cin : in std_logic;
        sum : out std_logic_vector (3 downto 0);
```

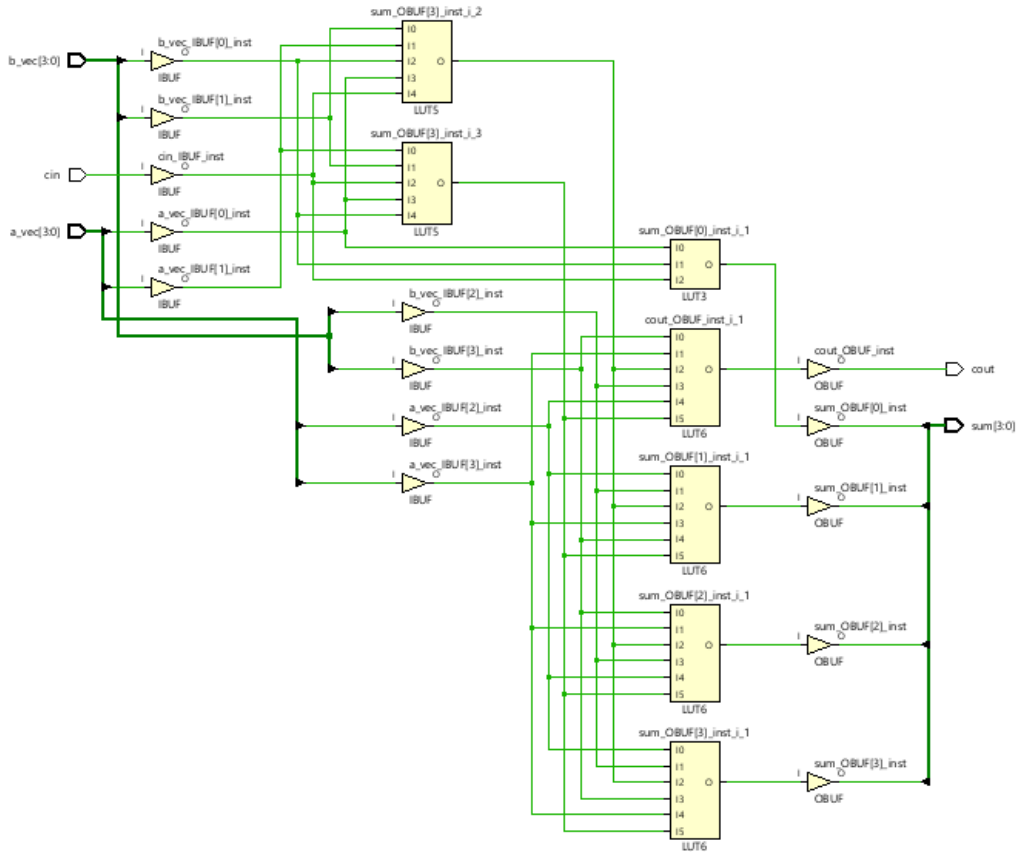
```
        cout : out std_logic
        );
end bcd_full_adder;


-- structural architecture
architecture Structural of bcd_full_adder is

    signal sum1: std_logic_vector (3 downto 0);
    signal cout1: std_logic;
    signal cout2 : std_logic;
    signal and1 : std_logic;
    signal and2 : std_logic;
    signal or1 : std_logic;
    signal add_in : std_logic_vector (3 downto 0);



    component parallel_4bit_adder is
    port(
        a_vec : in std_logic_vector(3 downto 0);
        b_vec : in std_logic_vector(3 downto 0);
        c_1 : in std_logic;
        sum_4 : out std_logic_vector(3 downto 0);
        carry_5 : out std_logic
    );
    end component;

     begin

        u1: parallel_4bit_adder port map
        (
            a_vec => a_vec,
            b_vec => b_vec,
            c_1 => cin,
            carry_5 => cout1,
            sum_4 => sum1

        );

        and1 <= sum1(3) and sum1(2);
        and2 <= sum1(3) and sum1(1);
        or1 <= cout1 or and1 or and2;
        add_in <= '0' & or1 & or1 & '0';

        u2: parallel_4bit_adder port map
        (
            a_vec => sum1,
            b_vec => add_in,
            c_1 => '0',
            carry_5 => cout2,
            sum_4 => sum

        );
```

```
        cout <= or1;

end Structural;
```

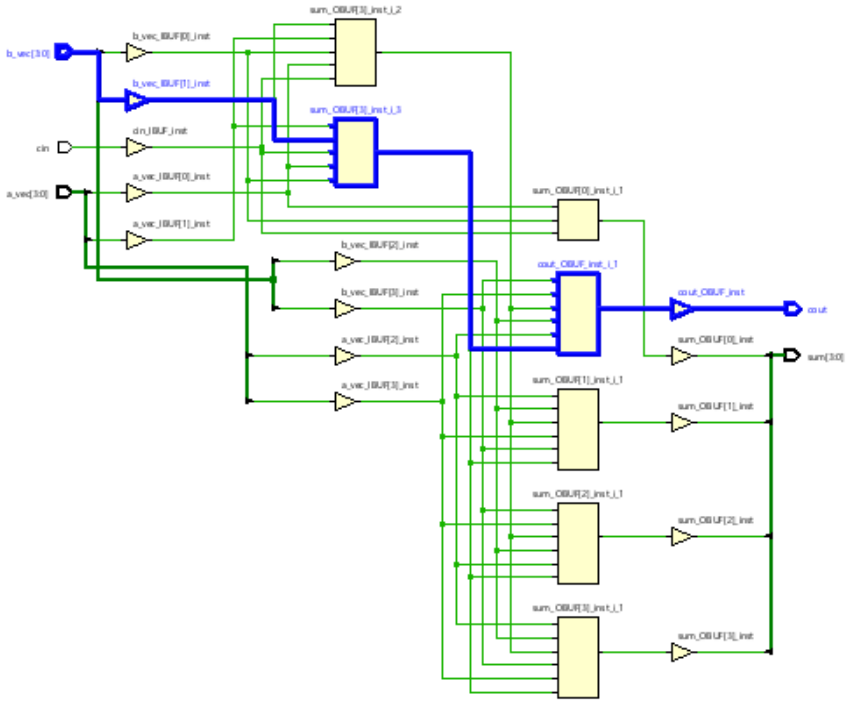
Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis παίρνουμε το εξής σχηματικό:



Χρησιμοποιώντας το timing summary για το κύκλωμα μας μπορούμε να βρούμε το κρίσιμο μονοπάτι του κυκλώματος μας, το οποίο φαίνεται παρακάτω:



Όπως φαίνεται παρακάτω η χρονική καθυστέρηση του critical path του κυκλώματος είναι 5.951 ns.

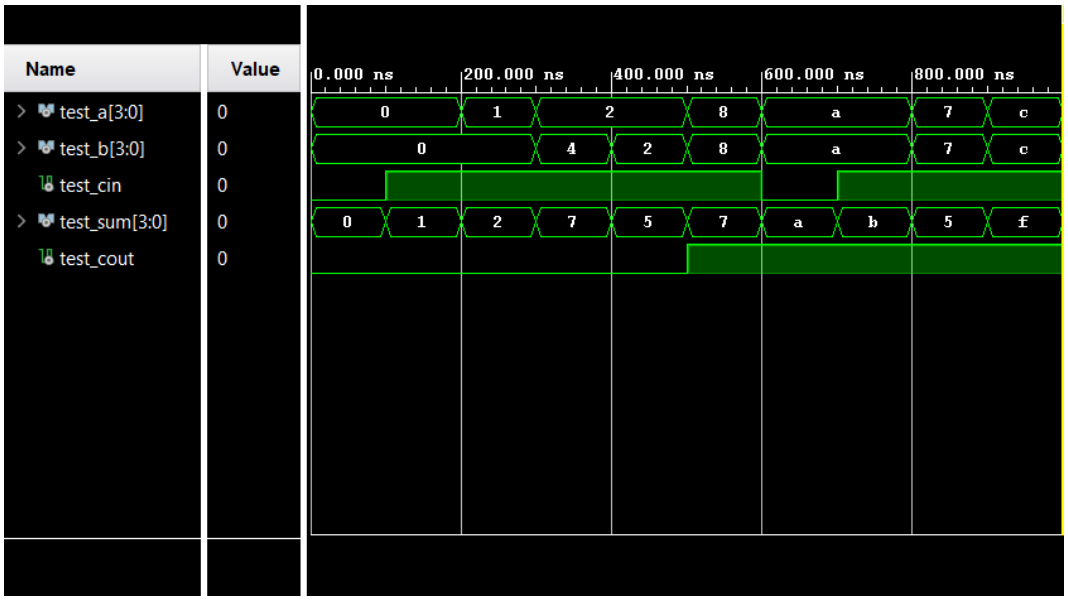
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	4	5	4	b_vec[1]	cout	5.951	3.878	2.072	∞	input port clock
Path 2	∞	4	5	4	b_vec[1]	sum[1]	5.951	3.878	2.072	∞	input port clock
Path 3	∞	4	5	4	b_vec[1]	sum[2]	5.951	3.878	2.072	∞	input port clock
Path 4	∞	4	5	4	b_vec[1]	sum[3]	5.951	3.878	2.072	∞	input port clock
Path 5	∞	3	4	3	cin	sum[0]	5.326	3.726	1.599	∞	input port clock

Για την προσομοίωση και τον έλεγχο της σωστής λειτουργίας του κυκλώματός μας χρησιμοποιούμε το παρακάτω testbench:

```
library ieee;

use ieee.std_logic_1164.all;

entity bcd_full_adder_tb is
end bcd_full_adder_tb;

architecture bcd_full_adder_tb_arch of bcd_full_adder_tb is

component bcd_full_adder is
    Port(
        a_vec : in std_logic_vector (3 downto 0);
        b_vec : 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;


-- stimulus signals


signal test_a   : std_logic_vector(3 downto 0);
signal test_b : std_logic_vector(3 downto 0);
signal test_cin : std_logic;
signal test_sum : std_logic_vector(3 downto 0);
signal test_cout : std_logic;

begin

uut: bcd_full_adder
    port map(
        a_vec => test_a,
        b_vec => test_b,
        sum => test_sum,
        cin => test_cin,
        cout => test_cout
    );

    testing : process

    begin

        test_a <= "0000";
        test_b <= "0000";
        test_cin <= '0';
        wait for 100 ns;

        test_a <= "0000";
        test_b <= "0000";
        test_cin  <= '1';
        wait for 100 ns;

        test_a <= "0001";
        test_b <= "0000";
        test_cin <= '1';
        wait for 100 ns;

        test_a <= "0010";
        test_b <= "0100";
        test_cin <= '1';
        wait for 100 ns;


        test_a <= "0010";
        test_b <= "0010";
        test_cin <= '1';
        wait for 100 ns;
```

```
        test_a <= "0100";
        test_b <= "0001";
        test_cin  <= '1';
        wait for 100 ns;

        test_a <= "0011";
        test_b <= "0011";
        test_cin <= '0';
        wait for 100 ns;


        test_a <= "0110";
        test_b <= "0011";
        test_cin <= '1';
        wait for 100 ns;


        test_a <= "0111";
        test_b <= "0111";
        test_cin <= '1';
        wait for 100 ns;

        test_a <= "0100";
        test_b <= "0111";
        test_cin <= '1';
        wait for 100 ns;

    end process;
end bcd_full_adder_tb_arch;
```



Παρατηρούμε ότι έχουμε ορθά αποτελέσματα για όλους τους συνδυασμούς εισόδων που δίνονται. Σημειώνεται ότι δεν είναι όλοοι οι πιθανοί συνδυασμοί εισόδων, αλλά είναι οι συνδυασμοί που πιθανόν να παρουσιαστούν σφάλματα, οπότε είναι αντιπροσωπευτικοί για την ορθότητα του κυκλώματος.

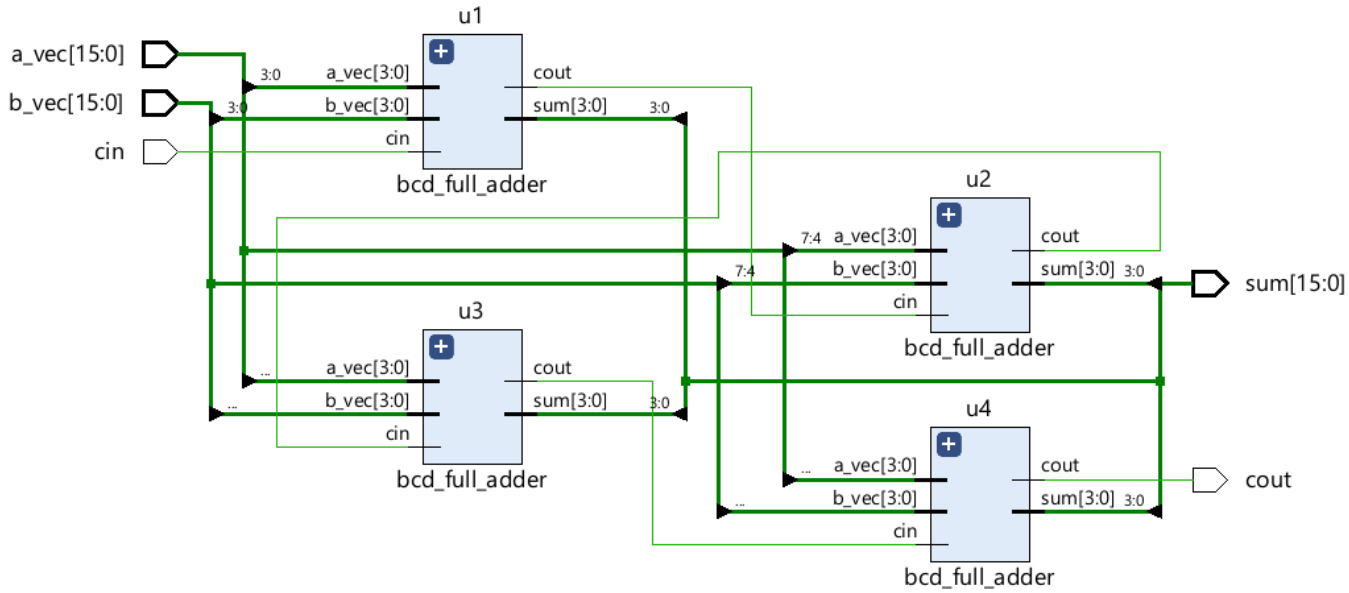
Θέμα 4: 4-BCD Parallel Adder σε περιγραφή structural

Ο κώδικας σε vhdl για την περιγραφή του bcd full adder και της αρχιτεκτονικής του σε structural vhdl είναι ο εξής:

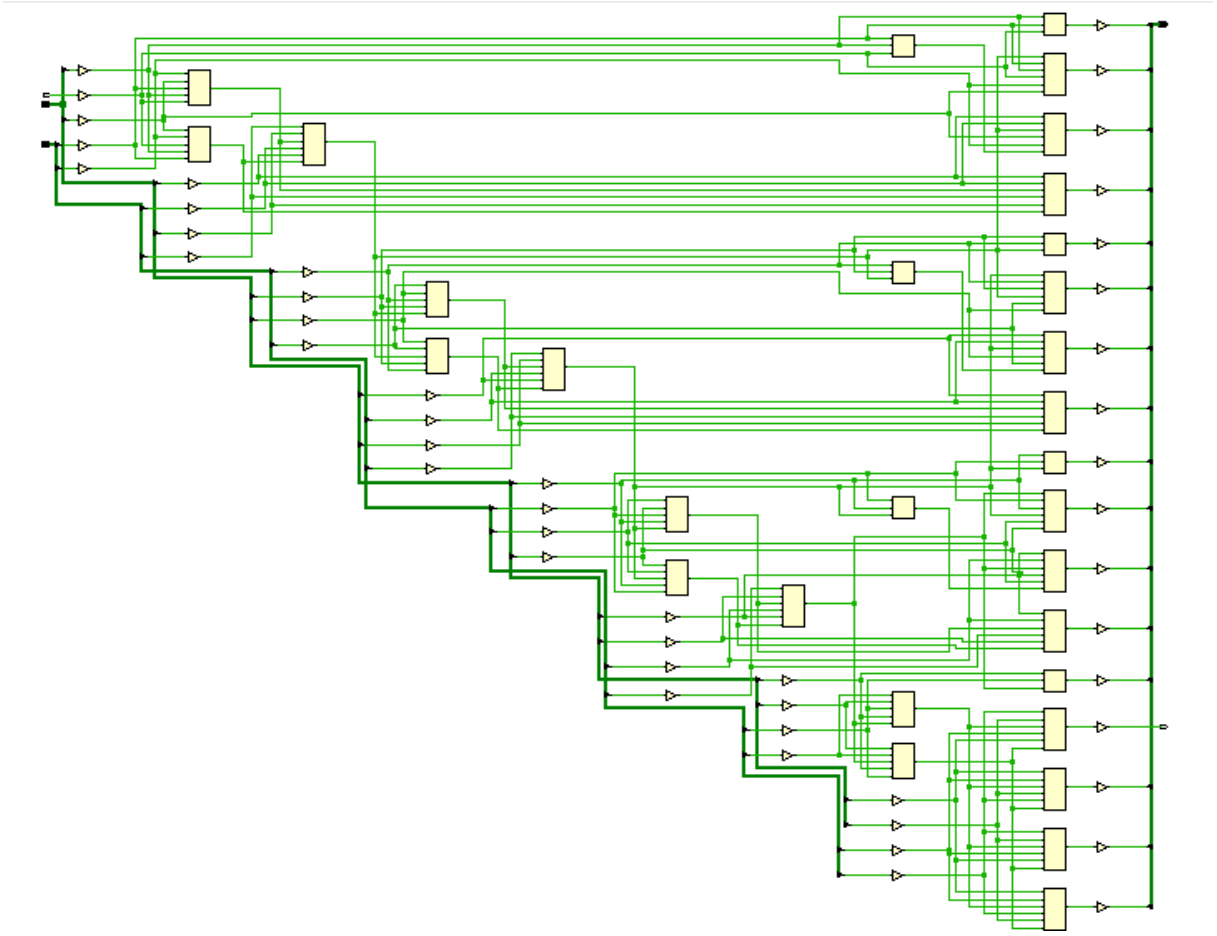
```
library ieee;

use ieee.std_logic_1164.all;

entity bcd_4bit_parallel_adder is
    Port(
        a_vec : in std_logic_vector (15 downto 0);
        b_vec : in std_logic_vector (15 downto 0);
        cin : in std_logic;
        sum : out std_logic_vector (15 downto 0);
        cout : out std_logic
        );
end bcd_4bit_parallel_adder;


-- structural architecture
architecture Structural of bcd_4bit_parallel_adder is

    signal cout1 : std_logic;
    signal cout2 : std_logic;
    signal cout3 : std_logic;


    component bcd_full_adder is
    port(
        a_vec : in std_logic_vector(3 downto 0);
        b_vec : 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;

     begin

        u1: bcd_full_adder port map
        (
            a_vec => a_vec(3 downto 0),
            b_vec => b_vec(3 downto 0),
            cin => cin,
            cout => cout1,
            sum => sum(3 downto 0)

        );

        u2: bcd_full_adder port map
        (
            a_vec => a_vec(7 downto 4),
            b_vec => b_vec(7 downto 4),
```

```
        cin => cout1,
        cout => cout2,
        sum => sum(7 downto 4)

    );

    u3: bcd_full_adder port map
    (
        a_vec => a_vec(11 downto 8),
        b_vec => b_vec(11 downto 8),
        cin => cout2,
        cout => cout3,
        sum => sum(11 downto 8)

    );

    u4: bcd_full_adder port map
    (
        a_vec => a_vec(15 downto 12),
        b_vec => b_vec(15 downto 12),
        cin => cout3,
        cout => cout,
        sum => sum(15 downto 12)

    );

end Structural;
```

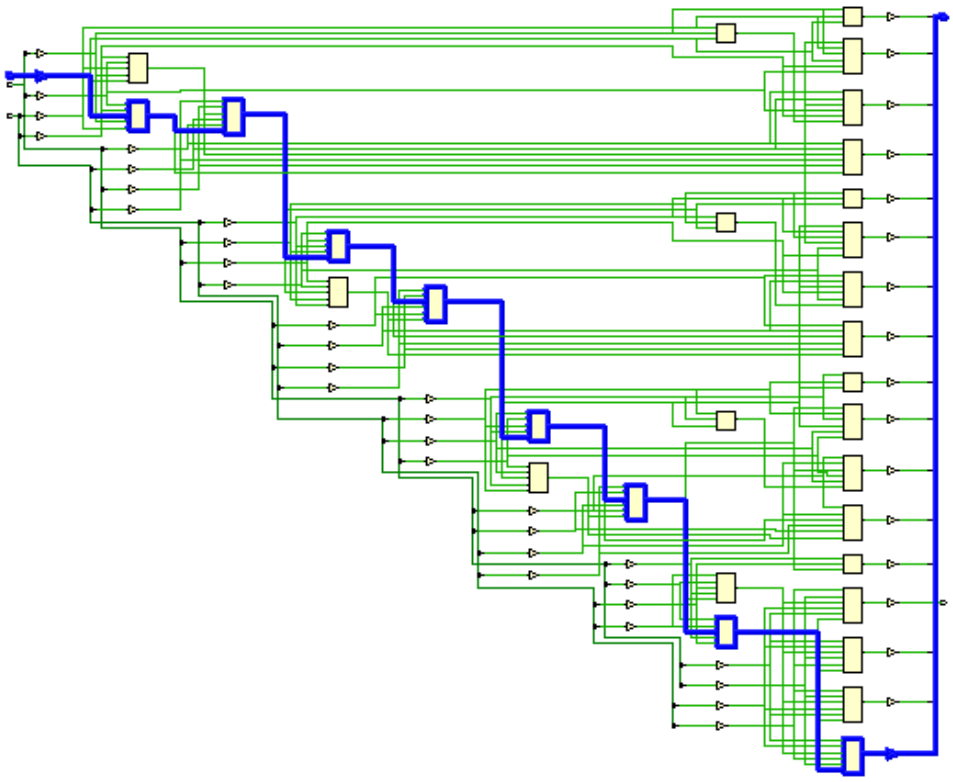
Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis παίρνουμε το εξής σχηματικό:



Χρησιμοποιώντας το timing summary για το κύκλωμα μας μπορούμε να βρούμε το κρίσιμο μονοπάτι του κυκλώματος μας, το οποίο φαίνεται παρακάτω:



Όπως φαίνεται παρακάτω η χρονική καθυστέρηση του critical path του κυκλώματος είναι 18.269 ns.

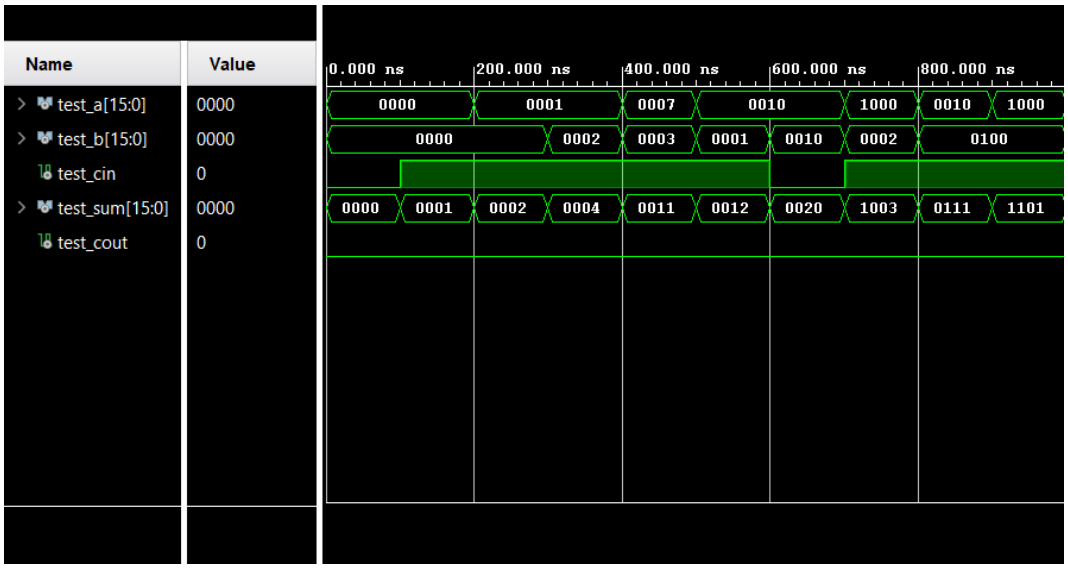
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	10	9	7	cin	sum[15]	18.269	5.280	12.989	∞	input port clock			0.000
Path 2	∞	10	9	7	cin	sum[14]	18.256	5.261	12.995	∞	input port clock			0.000
Path 3	∞	10	9	7	cin	cout	17.828	5.052	12.776	∞	input port clock			0.000
Path 4	∞	10	9	7	cin	sum[13]	16.592	5.041	11.552	∞	input port clock			0.000
Path 5	∞	9	8	7	cin	sum[12]	15.647	4.925	10.723	∞	input port clock			0.000
Path 6	∞	9	8	7	cin	sum[9]	15.418	4.892	10.526	∞	input port clock			0.000
Path 7	∞	9	8	7	cin	sum[10]	15.203	4.920	10.284	∞	input port clock			0.000
Path 8	∞	8	7	7	cin	sum[11]	14.546	4.796	9.749	∞	input port clock			0.000
Path 9	∞	7	6	7	cin	sum[8]	13.800	4.639	9.161	∞	input port clock			0.000
Path 10	∞	7	6	7	cin	sum[6]	13.238	4.424	8.814	∞	input port clock			0.000

Για την προσομοίωση και τον έλεγχο της σωστής λειτουργίας του κυκλώματός μας χρησιμοποιούμε το παρακάτω testbench:

```
library ieee;

use ieee.std_logic_1164.all;

entity bcd_4bit_parallel_adder_tb is
end bcd_4bit_parallel_adder_tb;

architecture  bcd_4bit_parallel_adder_tb_arch of bcd_4bit_parallel_adder_tb is
```

```
component bcd_4bit_parallel_adder is
    Port(
        a_vec : in std_logic_vector (15 downto 0);
        b_vec : in std_logic_vector (15 downto 0);
        cin : in std_logic;
        sum : out std_logic_vector (15 downto 0);
        cout : out std_logic
        );
end component;


-- stimulus signals

signal test_a   : std_logic_vector(15 downto 0);
signal test_b : std_logic_vector(15 downto 0);
signal test_cin : std_logic;
signal test_sum : std_logic_vector(15 downto 0);
signal test_cout : std_logic;

begin

uut: bcd_4bit_parallel_adder
    port map(
        a_vec => test_a,
        b_vec => test_b,
        sum => test_sum,
        cin => test_cin,
        cout => test_cout
    );

    testing : process

    begin

        test_a <= x"0000";
        test_b <= x"0000";
        test_cin <= '0';
        wait for 100 ns;

        test_a <= x"0000";
        test_b <= x"0000";
        test_cin  <= '1';
        wait for 100 ns;

        test_a <= x"0001";
        test_b <= x"0000";
        test_cin <= '1';
        wait for 100 ns;

        test_a <= x"0001";
        test_b <= x"0002";
        test_cin <= '1';
        wait for 100 ns;
```

```
        test_a <= x"0007";
        test_b <= x"0003";
        test_cin <= '1';
        wait for 100 ns;

        test_a <= x"0010";
        test_b <= x"0001";
        test_cin  <= '1';
        wait for 100 ns;

        test_a <= x"0010";
        test_b <= x"0010";
        test_cin <= '0';
        wait for 100 ns;

        test_a <= x"1000";
        test_b <= x"0002";
        test_cin <= '1';
        wait for 100 ns;

        test_a <= x"0010";
        test_b <= x"0100";
        test_cin <= '1';
        wait for 100 ns;

        test_a <= x"1000";
        test_b <= x"0100";
        test_cin <= '1';
        wait for 100 ns;

    end process;
end bcd_4bit_parallel_adder_tb_arch;
```



Παρατηρούμε ότι έχουμε ορθά αποτελέσματα για όλους τους συνδυασμούς εισόδων που δίνονται. Σημειώνεται ότι δεν είναι όλοοι οι πιθανοί συνδυασμοί εισόδων, αλλά είναι οι συνδυασμοί που πιθανόν να παρουσιαστούν σφάλματα, οπότε είναι αντιπροσωπευτικοί για την ορθότητα του κυκλώματος.