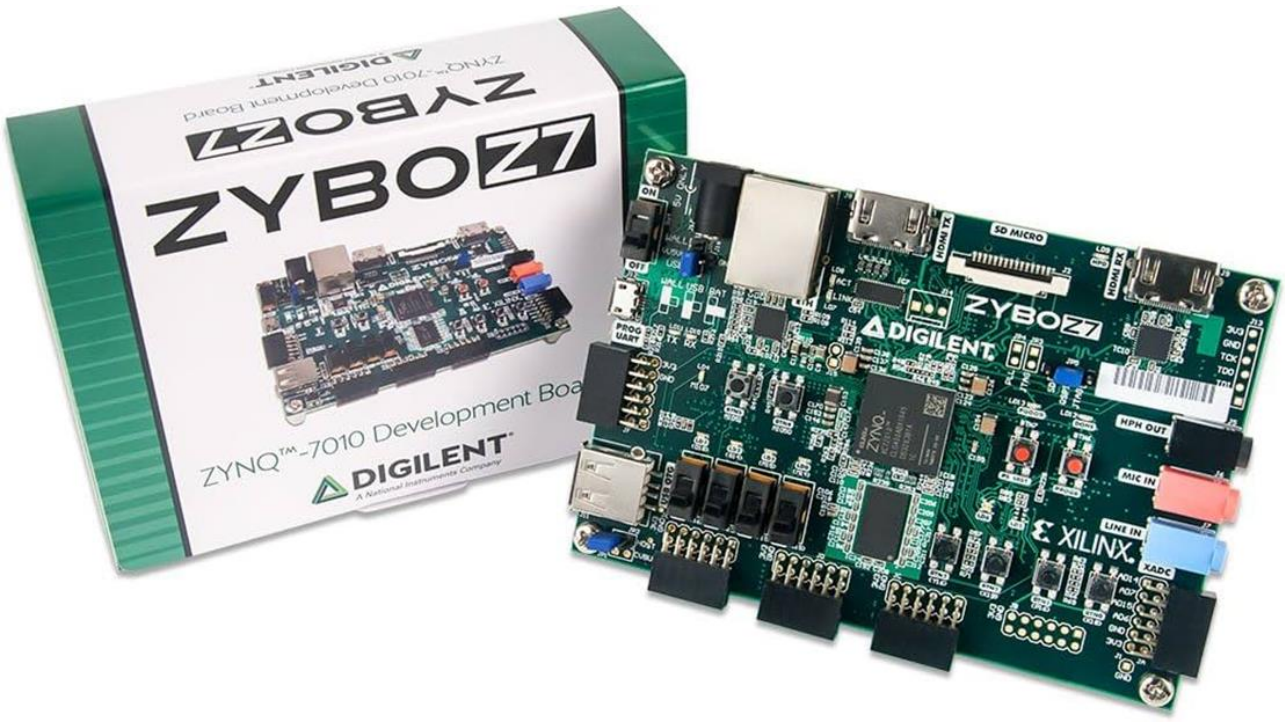
Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών



Μάθημα: Ψηφιακά Συστήματα VLSI

Θέμα: 2η Εργαστηριακή Άσκηση – Σχεδιασμός Αριθμητικών Μονάδων με Ιεραρχική Σχεδίαση

Εξάμηνο: 8ο

Ομάδα: 6

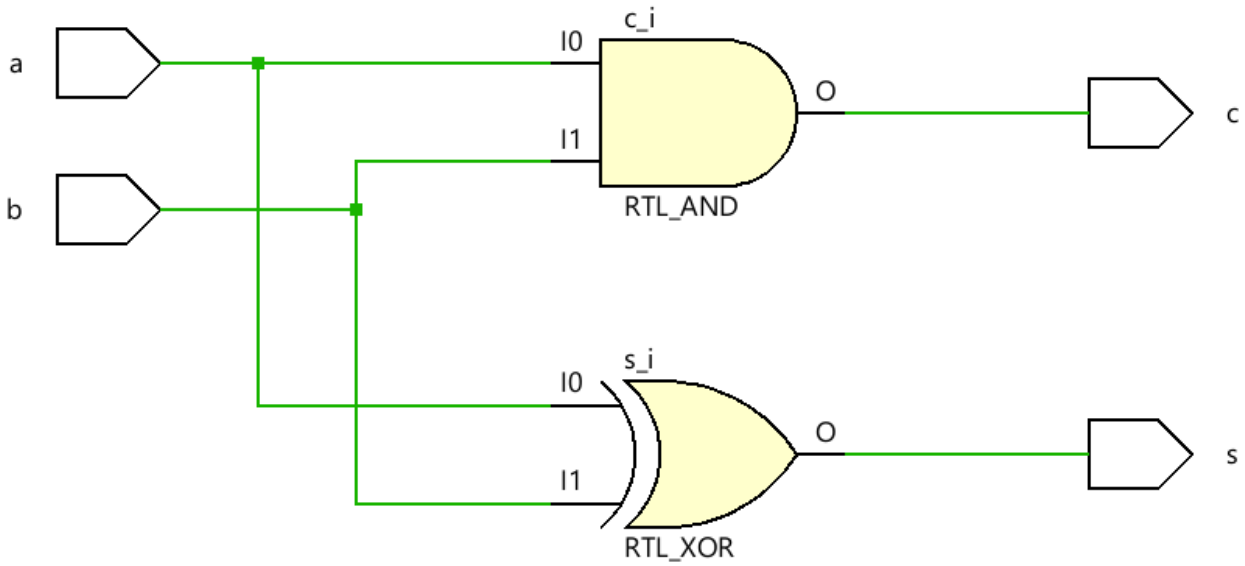
Συνεργάτες:

- Ακύλας Αντώνιος 03121152
- Κουμπιάς Ιωάννης-Χρυσοβαλάντης 03121053

Άσκηση 1

Στο πρώτο κομμάτι του εργαστηρίου σχεδιάσαμε έναν Ημιαθροιστή (Half Adder - HA) σε περιγραφή ροής δεδομένων (Dataflow). Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του κυκλώματος που προκύπτει:

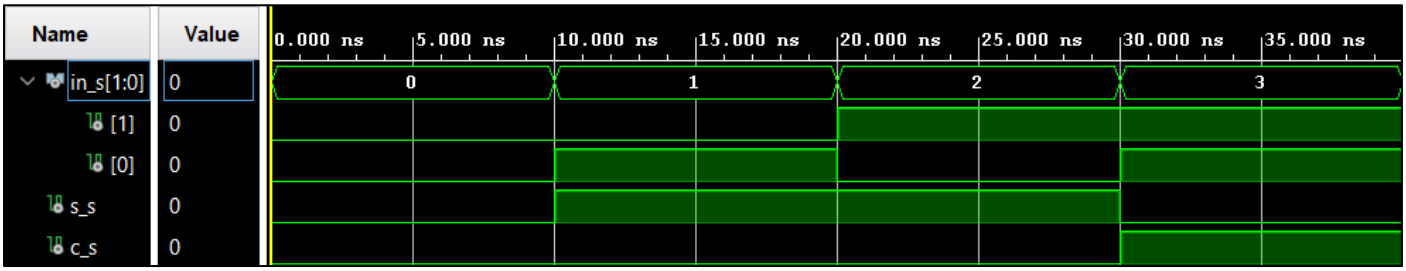
```
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
    port(
        a: in std_logic;
        b: in std_logic;
        s: out std_logic;
        c: out std_logic
    );
end half_adder;

architecture dataflow of half_adder is
begin
    s <= a xor b;
    c <= a and b;
end dataflow;
```



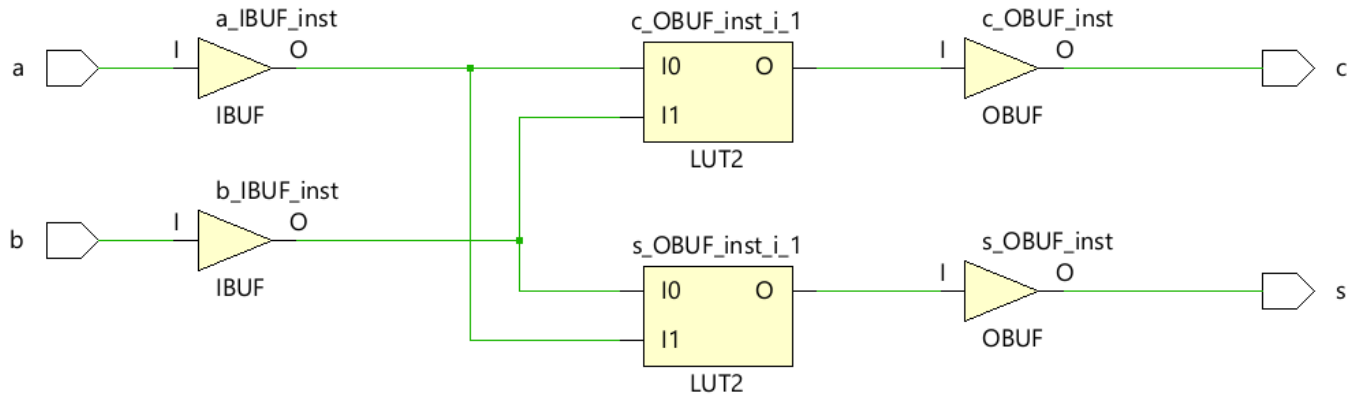
Στην συνέχεια δημιουργήσαμε κατάλληλο testbench που παράγει όλα τα διαφορετικά inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και τα αποτελέσματα του simulation:

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ha_tb is
end ha_tb;

architecture ha_tb_arch of ha_tb is
    component half_adder
        port(
            a: in std_logic;
            b: in std_logic;
            s: out std_logic;
```

```
            c: out std_logic
        );
    end component;
    signal in_s: std_logic_vector(1 downto 0);
    signal s_s,c_s: std_logic;
begin
    UUT: half_adder port map (in_s(0),in_s(1),s_s,c_s);
    testSequence: process
        begin
            in_s <= "00";
            for i in 0 to 3 loop
                wait for 10 ns;
                in_s <= in_s + 1;
            end loop;
    end process;
end ha_tb_arch;
```



Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ' επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:



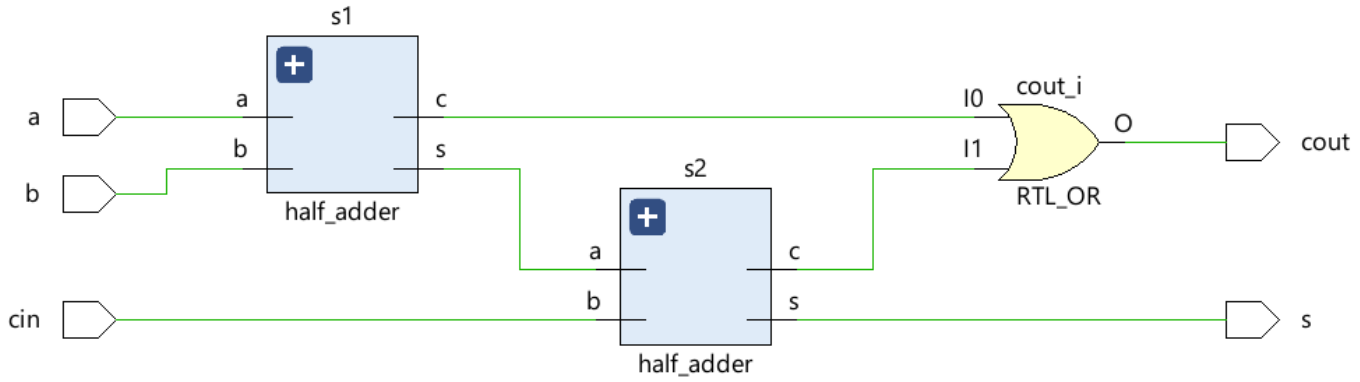
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	3	4	2	b	s	5.377	3.778	1.599
Path 2	∞	3	4	2	b	c	5.351	3.752	1.599

Το critical path της σύνθεσης φάινεται πρώτο στα Unconstrained Paths του timing report. Στις μετρήσεις συνυπολογίζονται τόσο η καθυστέρηση λόγω λογικής (Logic Delay) όσο και η καθυστέρηση λόγω του routing των συνδέσεων στο FPGA (Net Delay). Παρατηρούμε πως το κρίσιμο μονοπάτι είναι από το input b προς το output s με μικρή διαφορά από το b – c (που οφείλεται στη λογική), γεγονός που αιτιολογείται από την ομοιότητα των δύο μονοπατιών.

Άσκηση 2

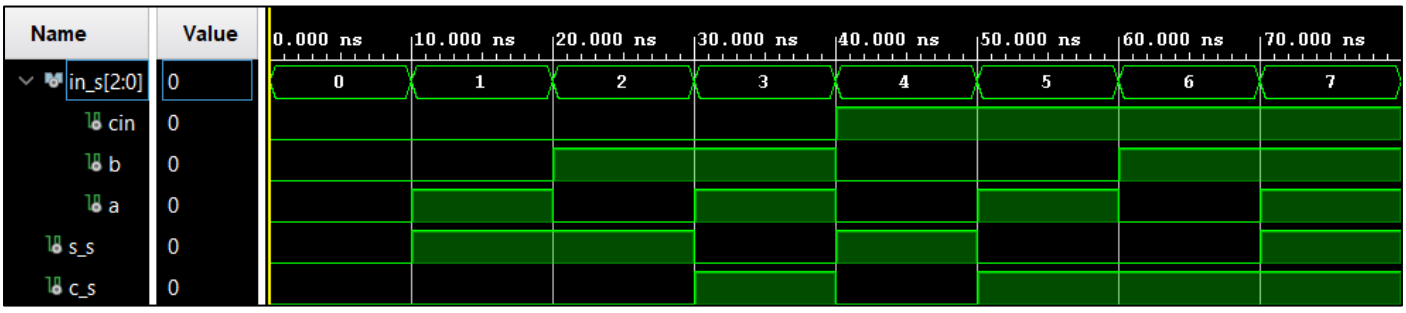
Στο δεύτερο κομμάτι του εργαστηρίου σχεδιάσαμε έναν Πλήρη Αθροιστή (Full Adder - FA) σε περιγραφή δομής (Structural) βασιζόμενοι στην δομική μονάδα της προηγούμενης άσκησης. Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του κυκλώματος που προκύπτει:

```
library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
    port(
        a: in std_logic;
        b: in std_logic;
        cin: in std_logic;
        s: out std_logic;
        cout: out std_logic
    );
end full_adder;

architecture structural of full_adder is
    component half_adder
        port(
            a: in std_logic;
            b: in std_logic;
            s: out std_logic;
            c: out std_logic
        );
    end component;
    signal sum,c1,c2: std_logic;
begin
    s1: half_adder port map(
        a => a,
        b => b,
        s => sum,
        c => c1
    );
    s2: half_adder port map(
        a => sum,
        b => cin,
        s => s,
        c => c2
    );
    cout <= c1 or c2;
end structural;
```

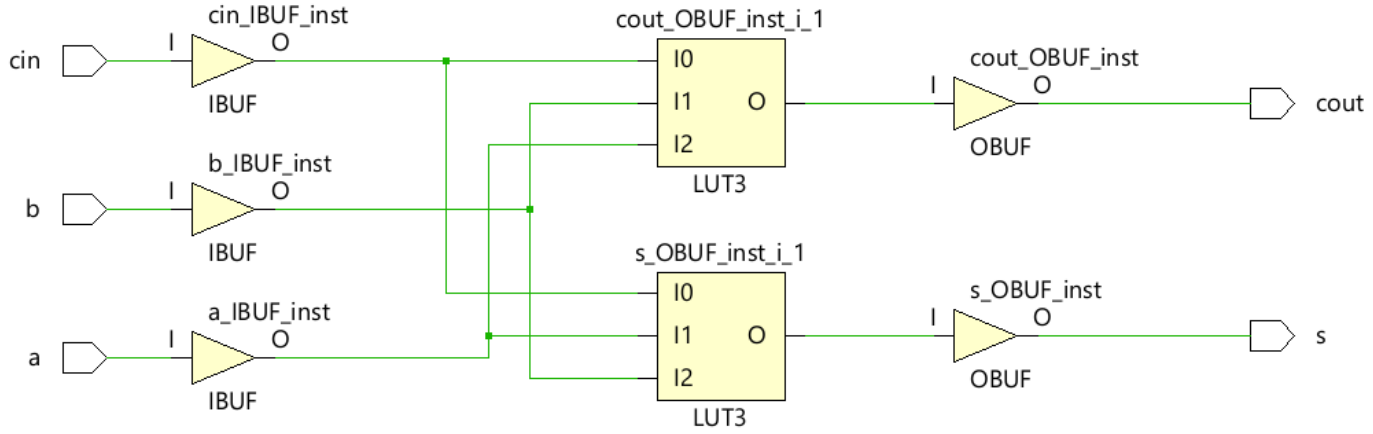


Στην συνέχεια δημιουργήσαμε κατάλληλο testbench που παράγει όλα τα διαφορετικά inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και τα αποτελέσματα του simulation:

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity fa_tb is
end fa_tb;

architecture fa_tb_arch of fa_tb is
    component full_adder
        port(
            a: in std_logic;
            b: in std_logic;
            cin: in std_logic;
            s: out std_logic;
            cout: out std_logic
        );
    end component;
    signal in_s: std_logic_vector(2 downto 0);
    signal s_s,c_s: std_logic;
begin
    UUT: full_adder port map (in_s(0),in_s(1),in_s(2),s_s,c_s);
    testSequence: process
        begin
            in_s <= "000";
            for i in 0 to 7 loop
                wait for 10 ns;
                in_s <= in_s + 1;
            end loop;
    end process;
end fa_tb_arch;
```



Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ' επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:



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

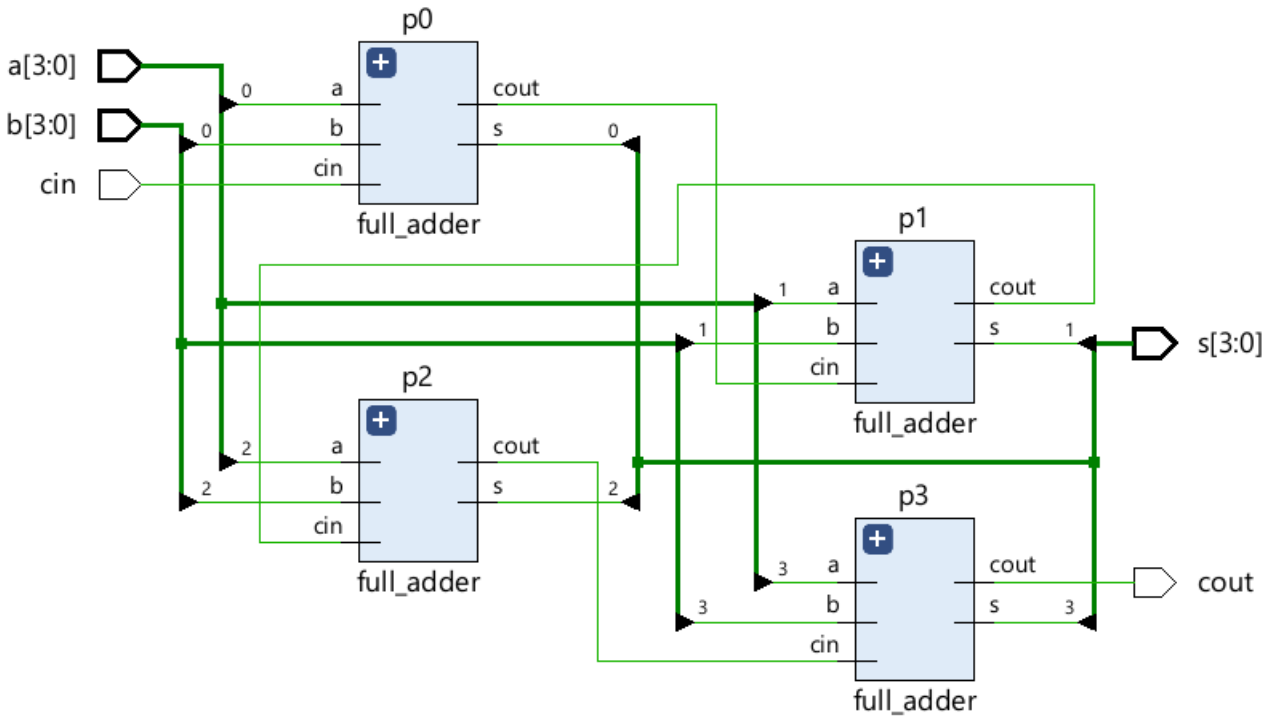
Παρατηρούμε πως το κρίσιμο μονοπάτι είναι από το input a προς το output s με μικρή διαφορά από το a – cout (που οφείλεται στη λογική), γεγονός που αιτιολογείται από την ομοιότητα των δύο μονοπατιών.

Άσκηση 3

Στο τρίτο κομμάτι του εργαστηρίου σχεδιάσαμε έναν Παράλληλο Αθροιστή των 4 bits (4-bit Parallel Adder – 4-bit PA) σε περιγραφή δομής (Structural) βασιζόμενοι στην δομική μονάδα της προηγούμενης άσκησης. Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του κυκλώματος που προκύπτει:

```
library ieee;
use ieee.std_logic_1164.all;

entity par_adder4 is
    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 par_adder4;

architecture structural of par_adder4 is
    component full_adder
        port(
            a: in std_logic;
            b: in std_logic;
            cin: in std_logic;
            s: out std_logic;
            cout: out std_logic
        );
    end component;
    signal c: std_logic_vector(3 downto 1);
begin
    p0: full_adder port map(a(0),b(0),cin,s(0),c(1));
    p1: full_adder port map(a(1),b(1),c(1),s(1),c(2));
```

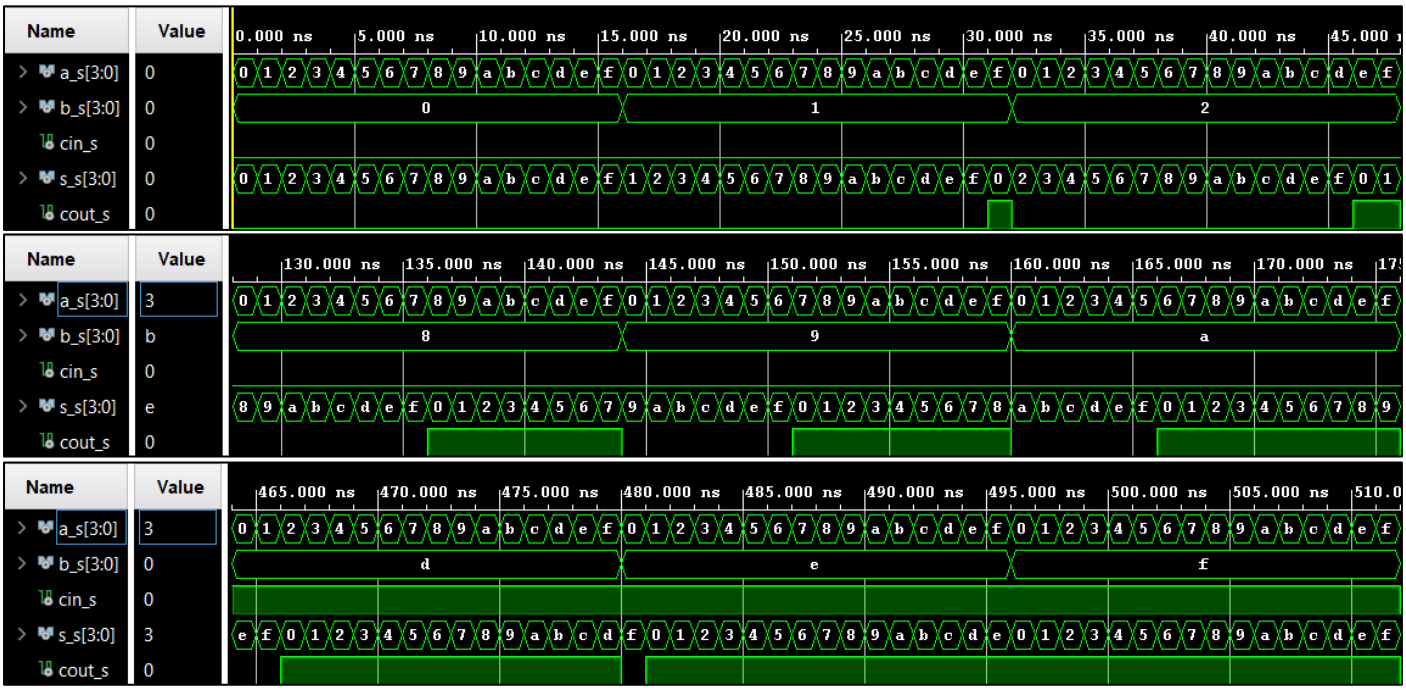
```
    p2: full_adder port map(a(2),b(2),c(2),s(2),c(3));
    p3: full_adder port map(a(3),b(3),c(3),s(3),cout);
end structural;
```



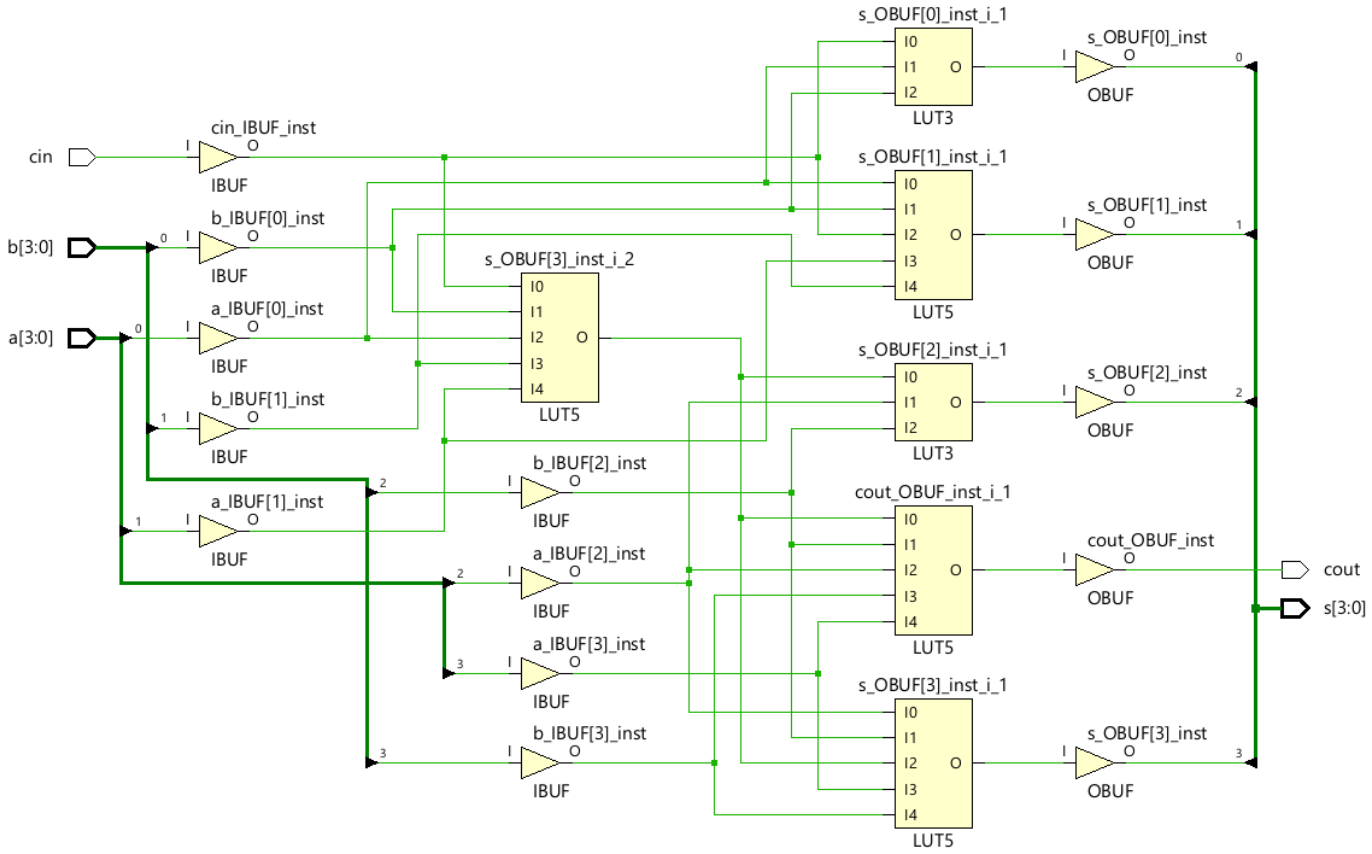
Στην συνέχεια δημιουργήσαμε κατάλληλο testbench που παράγει όλα τα διαφορετικά inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και κάποια αντιπροσωπευτικά αποτελέσματα του simulation:

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity pa4_tb is
end pa4_tb;

architecture pa4_tb_arch of pa4_tb is
    component par_adder4
        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;
    signal a_s: std_logic_vector(3 downto 0);
    signal b_s: std_logic_vector(3 downto 0);
    signal cin_s: std_logic;
    signal s_s: std_logic_vector(3 downto 0);
    signal cout_s: std_logic;
begin
    UUT: par_adder4 port map (a_s,b_s,cin_s,s_s,cout_s);
    testSequence: process
        begin
            a_s <= "0000";
```

```
            b_s <= "0000";

            cin_s <= '0';
            for i in 0 to 15 loop
                for j in 0 to 15 loop
                    wait for 1 ns;
                    a_s <= a_s + 1;
                end loop;
                b_s <= b_s + 1;
            end loop;

            cin_s <= '1';
            for i in 0 to 15 loop
                for j in 0 to 15 loop
                    wait for 1 ns;
                    a_s <= a_s + 1;
                end loop;
                b_s <= b_s + 1;
            end loop;
    end process;
end pa4_tb_arch;
```



Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ' επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:



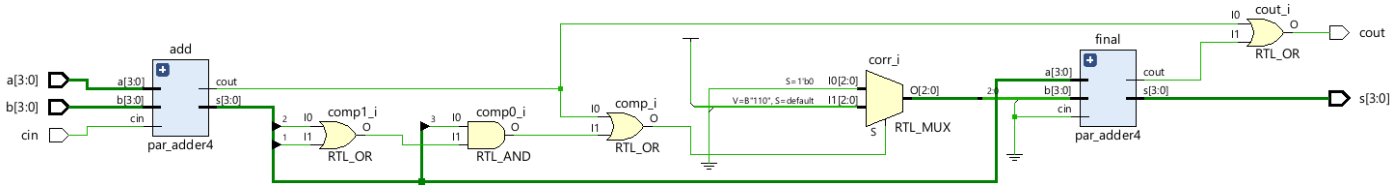
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	4	5	3	b[0]	cout	5.970	3.904	2.066
Path 2	∞	4	5	3	b[0]	s[2]	5.970	3.904	2.066
Path 3	∞	4	5	3	b[0]	s[3]	5.964	3.898	2.066
Path 4	∞	3	4	3	b[0]	s[0]	5.351	3.752	1.599
Path 5	∞	3	4	2	b[1]	s[1]	5.351	3.752	1.599

Παρατηρούμε πως τα κρίσιμα μονοπάτια είναι τα b[0] – cout και b[0] – s[2] με μικρή διαφορά από το b[0] – s[3] (οφείλεται στη λογική). Από την άλλη τα μονοπάτια b[0] – s[0] και b[1] – s[1] φαίνεται να έχουν μικρότερες καθυστερήσεις τόσο στη λογική όσο και στη δικτύωσή τους.

Άσκηση 4

Στο τέταρτο κομμάτι του εργαστηρίου σχεδιάσαμε έναν BCD Πλήρη Αθροιστή (BCD Full Adder – BCD FA) σε περιγραφή δομής (Structural). Η υλοποίηση βασίζεται τόσο στην δομική μονάδα του 4-bit Parallel Adder της προηγούμενης άσκησης όσο και σε επιπλέον λογική για την μετατροπή των αποτελεσμάτων της άθροισης σε BCD. Συγκεκριμένα εκτελούμε κανονικά την δυαδική άθροιση και στην συνέχεια αν το αποτέλεσμα είναι μεγαλύτερο του 10 τότε προσθέτουμε το 0110 (δυαδική αναπαράσταση του συμπληρώματος ως προς 2 του 10 – αφαίρεση) αλλιώς το 0000 (δηλαδή αφήνουμε το αποτέλεσμα όπως έχει). Τέλος έχουμε κρατούμενο εάν προκύψει κρατούμενο από κάποια εκ των προσθέσεων. Ιδιαίτερο ενδιαφέρον παρουσιάζει ο τρόπος ελέγχου ο οποίος βασίζεται στην παρατήρηση πως το αποτέλεσμα της πρώτης πρόσθεσης είναι μεγαλύτερο ή ίσο του 10 ανν προκύπτει κρατούμενο ή το 3ο bit είναι 1 και τουλάχιστον ένα από τα bit 2 ή 1 είναι 1. Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του κυκλώματος που προκύπτει:

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity bcd_fa is
    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 bcd_fa;

architecture structural of bcd_fa is
    component par_adder4
        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;
    signal s_inter: std_logic_vector(3 downto 0);
    signal corr: std_logic_vector(3 downto 0);
    signal c_inter,c_bcd,comp: std_logic;
begin
    add: par_adder4 port map(a,b,cin,s_inter,c_inter);
    comp <= c_inter or (s_inter(3) and (s_inter(2) or s_inter(1)));
    with comp select corr <=
        "0000" when '0',
        "0110" when others;
    final: par_adder4 port map(s_inter,corr,'0',s,c_bcd);
    cout <= c_inter or c_bcd;
end structural;
```

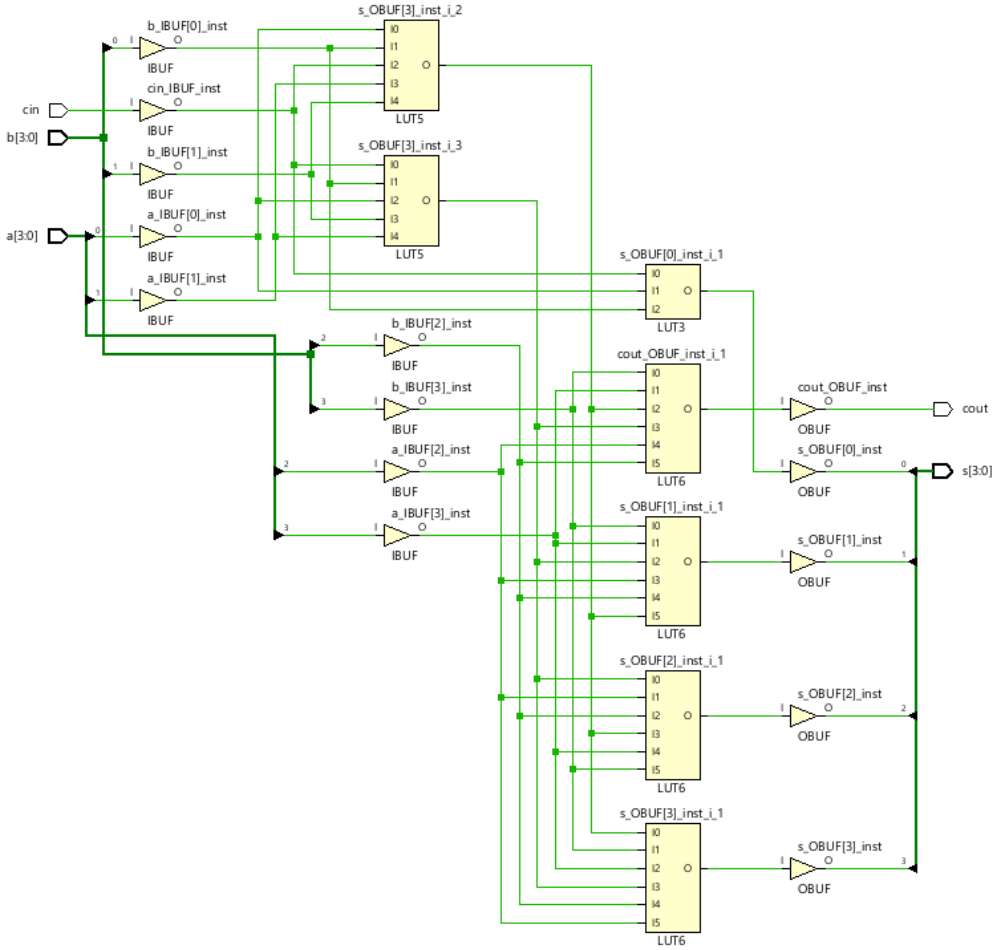


Στην συνέχεια δημιουργήσαμε κατάλληλο testbench που παράγει όλα τα διαφορετικά inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και αντιπροσωπευτικά αποτελέσματα του simulation:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity bcd_fa_tb is
end bcd_fa_tb;
```

```
architecture bcd_fa_tb_arch of bcd_fa_tb is
    component bcd_fa
        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;
    signal a_s: std_logic_vector(3 downto 0);
    signal b_s: std_logic_vector(3 downto 0);
    signal cin_s: std_logic;
    signal s_s: std_logic_vector(3 downto 0);
    signal cout_s: std_logic;
begin
    UUT: bcd_fa port map (a_s,b_s,cin_s,s_s,cout_s);
    testSequence: process
        begin
            b_s <= "0000";
            cin_s <= '0';
            for i in 0 to 9 loop
                a_s <= "0000";
                for j in 0 to 9 loop
                    wait for 1 ns;
                    a_s <= a_s + 1;
                end loop;
                b_s <= b_s + 1;
            end loop;

            b_s <= "0000";
            cin_s <= '1';
            for i in 0 to 9 loop
                a_s <= "0000";
                for j in 0 to 9 loop
                    wait for 1 ns;
                    a_s <= a_s + 1;
                end loop;
                b_s <= b_s + 1;
            end loop;
    end process;
end bcd_fa_tb_arch;
```

Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ' επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:



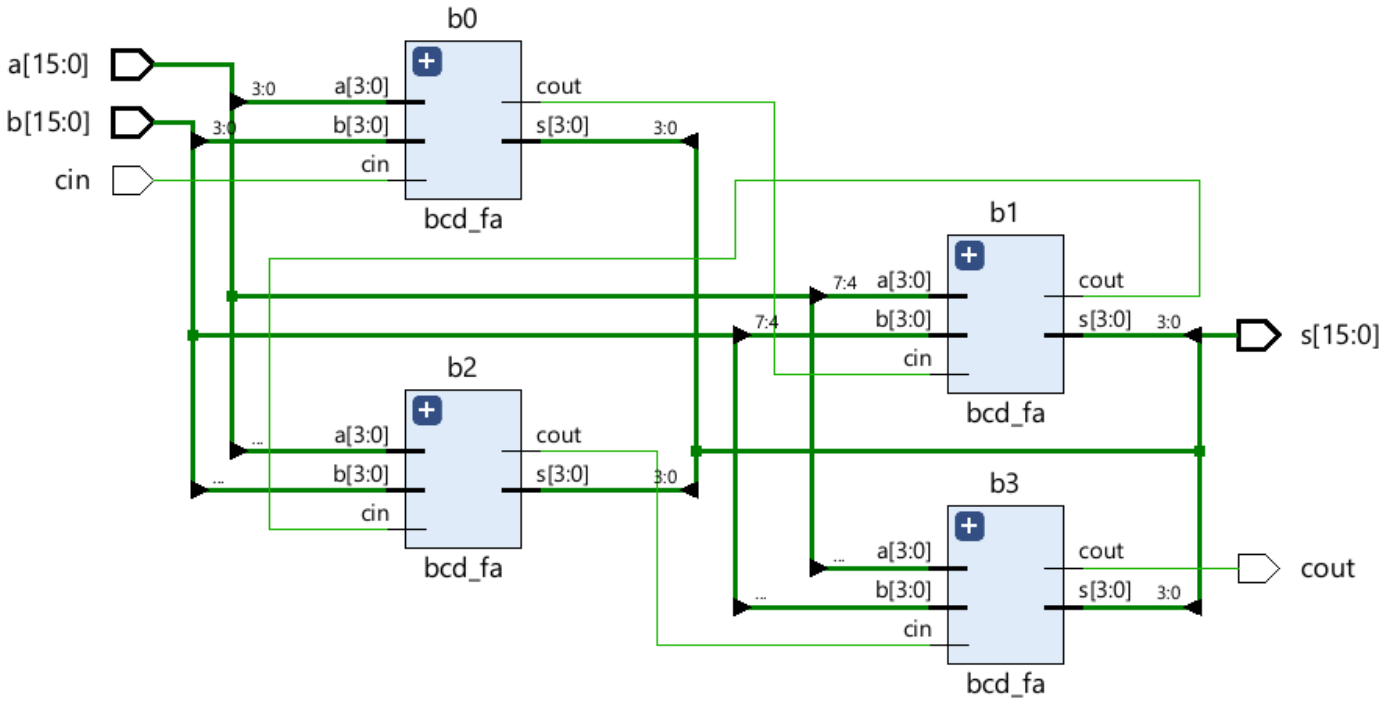
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	4	5	4	b[0]	cout	5.976	3.904	2.072
Path 2	∞	4	5	4	b[0]	s[3]	5.976	3.904	2.072
Path 3	∞	4	5	4	b[1]	s[1]	5.948	3.876	2.072
Path 4	∞	4	5	4	b[1]	s[2]	5.948	3.876	2.072
Path 5	∞	3	4	3	b[0]	s[0]	5.351	3.752	1.599

Παρατηρούμε πως τα κρίσιμα μονοπάτια είναι τα b[0] – cout και b[0] – s[3] με μικρή διαφορά από τα b[1] – s[1] και b[1] – s[2] (οφείλεται στη λογική). Από την άλλη το μονοπάτι b[0] – s[0] φαίνεται να έχει μικρότερες καθυστερήσεις τόσο στη λογική όσο και στη δικτύωσή του.

Άσκηση 5

Στο πέμπτο κομμάτι του εργαστηρίου σχεδιάσαμε έναν Παράλληλο BCD Αθροιστή των 4 ψηφίων (4-BCD Parallel Adder – 4-BCD PA) σε περιγραφή δομής (Structural) βασιζόμενοι στην δομική μονάδα της προηγούμενης άσκησης. Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του κυκλώματος που προκύπτει:

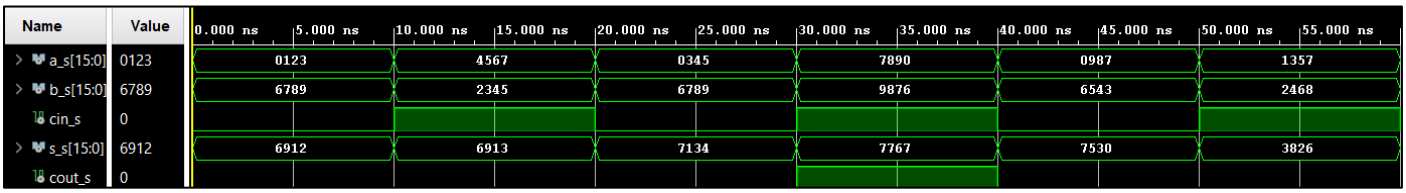
```
library ieee;
use ieee.std_logic_1164.all;

entity bcd4 is
    port(
        a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        cin: in std_logic;
        s: out std_logic_vector(15 downto 0);
        cout: out std_logic
    );
end bcd4;

architecture structural of bcd4 is
    component bcd_fa
        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;
    signal c: std_logic_vector(3 downto 1);
begin
    b0: bcd_fa port map(a(3 downto 0),b(3 downto 0),cin,s(3 downto 0),c(1));
    b1: bcd_fa port map(a(7 downto 4),b(7 downto 4),c(1),s(7 downto 4),c(2));
    b2: bcd_fa port map(a(11 downto 8),b(11 downto 8),c(2),s(11 downto 8),c(3));
    b3: bcd_fa port map(a(15 downto 12),b(15 downto 12),c(3),s(15 downto 12),cout);
end structural;
```



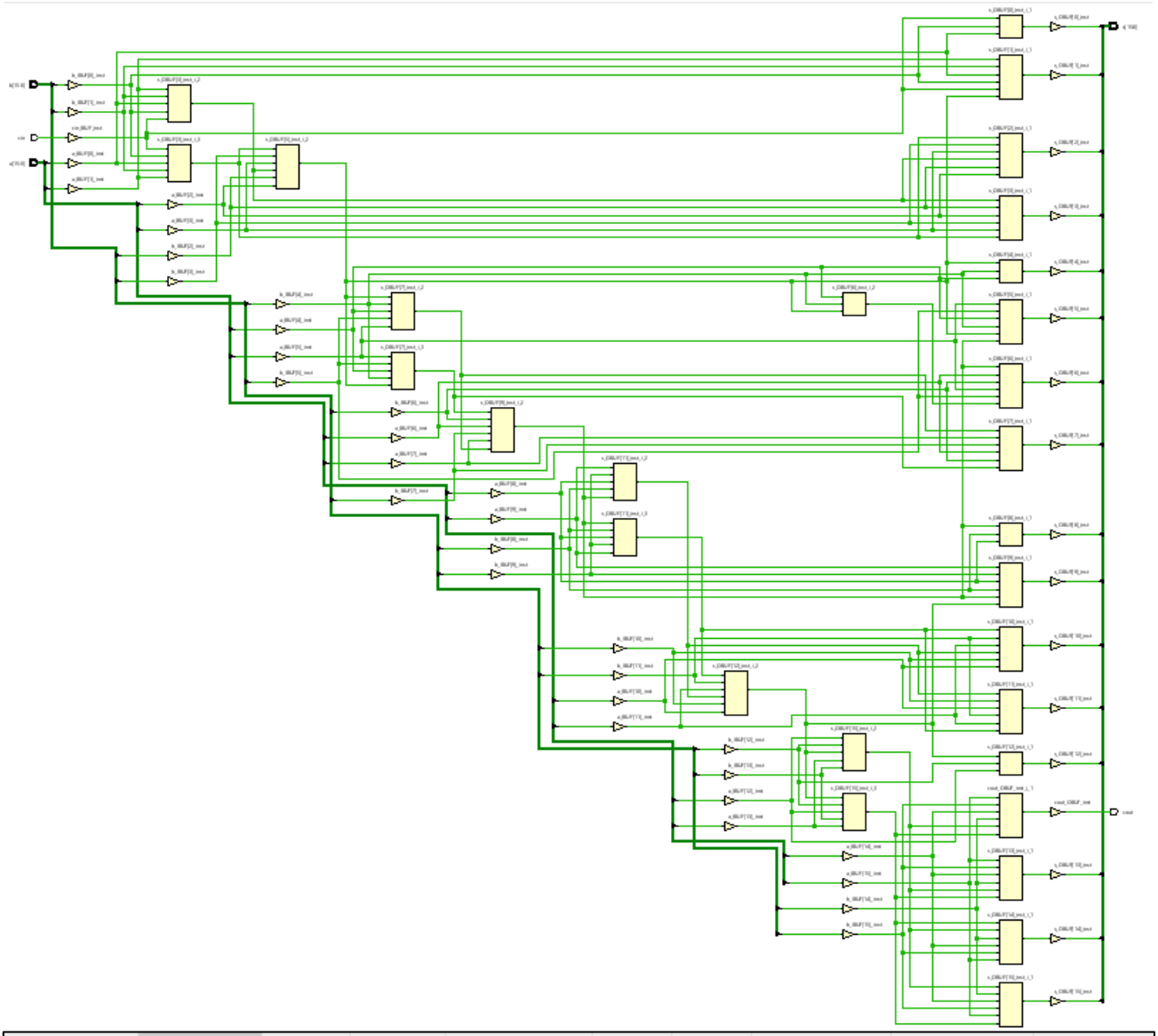
Στην συνέχεια δημιουργήσαμε κατάλληλο αντιπροσωπευτικό testbench που παράγει 6 τυχαία inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και τα αποτελέσματα του simulation:

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity bcd4_tb is
end bcd4_tb;

architecture bcd4_tb_arch of bcd4_tb is
    component bcd4
        port(
            a: in std_logic_vector(15 downto 0);
            b: in std_logic_vector(15 downto 0);
            cin: in std_logic;
            s: out std_logic_vector(15 downto 0);
            cout: out std_logic
        );
    end component;
    signal a_s: std_logic_vector(15 downto 0);
    signal b_s: std_logic_vector(15 downto 0);
    signal cin_s: std_logic;
    signal s_s: std_logic_vector(15 downto 0);
    signal cout_s: std_logic;
begin
    UUT: bcd4 port map (a_s,b_s,cin_s,s_s,cout_s);
    testSequence: process
        begin
            cin_s <= '0';
            a_s <= X"0123";
            b_s <= X"6789";
```

```
            wait for 10 ns;

            cin_s <= '1';
            a_s <= X"4567";
            b_s <= X"2345";

            wait for 10 ns;

            cin_s <= '0';
            a_s <= X"0345";
            b_s <= X"6789";

            wait for 10 ns;

            cin_s <= '1';
            a_s <= X"7890";
            b_s <= X"9876";

            wait for 10 ns;

            cin_s <= '0';
            a_s <= X"0987";
            b_s <= X"6543";

            wait for 10 ns;

            cin_s <= '1';
            a_s <= X"1357";
            b_s <= X"2468";

            wait for 10 ns;
    end process;
end bcd4_tb_arch;
```



Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ' επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:



Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	10	11	6	cin	s[14]	9.521	4.620	4.901
Path 2	∞	10	11	6	cin	cout	9.515	4.614	4.901
Path 3	∞	10	11	6	cin	s[13]	9.515	4.614	4.901
Path 4	∞	10	11	6	cin	s[15]	9.515	4.614	4.901
Path 5	∞	9	10	6	cin	s[12]	8.924	4.496	4.428
Path 6	∞	9	10	6	cin	s[9]	8.924	4.496	4.428
Path 7	∞	8	9	6	cin	s[10]	8.327	4.372	3.955
Path 8	∞	8	9	6	cin	s[11]	8.321	4.366	3.955
Path 9	∞	7	8	6	cin	s[5]	7.736	4.248	3.488
Path 10	∞	7	8	6	cin	s[6]	7.736	4.248	3.488

Παρατηρούμε πως το κρίσιμο μονοπάτι είναι από το cin προς το s[14] με αρκετή καθυστέρηση τόσο εξαιτίας της λογικής όσο και του τρόπου δικτύωσης, σε σχέση με τα υπόλοιπα μονοπάτια του πίνακα μετρήσεων.