ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σ.Η.Μ.Μ.Υ.
ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ VLSI

2η Εργαστηριακή Άσκηση

Ομάδα: **B13**

Στοιχεία φοιτητών:

Γκανάς Γεώργιος (**031 16 095**)
Περδικούρης Ορφέας (**031 15 081**)
Αϊβασιλιώτης Παναγιώτης (**031 16 176**)

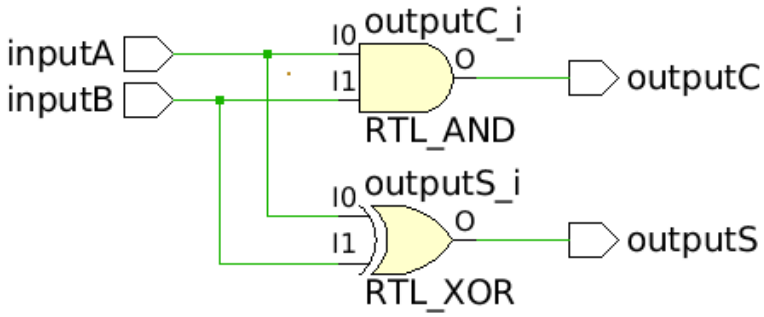
Ζήτημα 1: Ημιαθροιστής

Ακολουθεί υλοποίηση σε VHDL ημιαθροιστή σε περιγραφή ροής δεδομένων.

Κώδικας

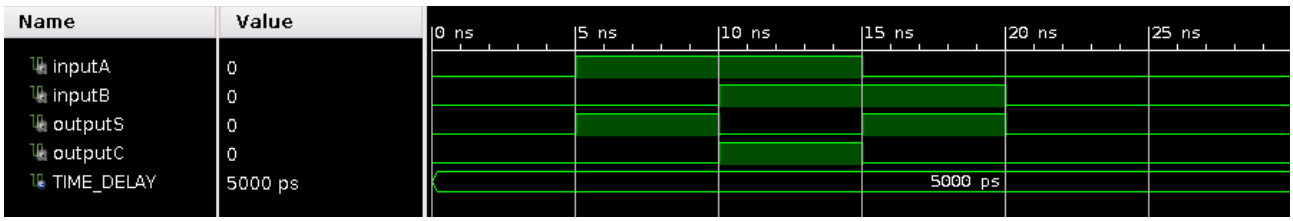
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity half_adder is
    Port (inputA  : in  STD_LOGIC;
          inputB  : in  STD_LOGIC;
          outputS : out STD_LOGIC;
          outputC : out STD_LOGIC);
end half_adder;

architecture HA_dataflow of half_adder is

begin
    outputS <= inputA xor inputB;
    outputC <= inputA and inputB;
end HA_dataflow;
```

Διάγραμμα



Προσομοίωση

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

entity half_adder_tb is
end half_adder_tb;

architecture Behavioral of half_adder_tb is

    component half_adder is
        port(
        inputA  : in STD_LOGIC;
        inputB  : in STD_LOGIC;
        outputS : out STD_LOGIC;
        outputC : out STD_LOGIC
        );
    end component;

    signal inputA  : STD_LOGIC;
    signal inputB  : STD_LOGIC;
    signal outputS : STD_LOGIC;
    signal outputC : STD_LOGIC;

    constant TIME_DELAY : time := 5 ns;
begin
    uut : half_adder
     port map(
        inputA,
        inputB,
        outputS,
        outputC
     );
    stimulus : process
    begin
        inputA <= '0';
        inputB <= '0';
        wait for TIME_DELAY;
        inputA <= '1';
        wait for TIME_DELAY;
        inputB <= '1';
        wait for TIME_DELAY;
        inputA <= '0';
        wait for TIME_DELAY;
        inputB <= '0';
        wait;
    end process;

end Behavioral;
```



Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος είναι το μονοπάτι inputA – outputS, με συνολική καθυστέρηση 4.641 ns.

Κατανάλωση Πόρων

Resource	Utilization	Available	Utilization...
Slice LUTs	1	17600	0.01
IO	4	102	3.92

Ζήτημα 2: Πλήρης Αθροιστής

Για το ζήτημα αυτό υλοποιήθηκαν πλήρεις αθροιστές με τέσσερις διαφορετικούς τρόπους: a) συνδυαστικό και ακολουθιακό κύκλωμα με περιγραφή δομής και b) συνδυαστικό και ακολουθιακό κύκλωμα με περιγραφή συμπεριφοράς.

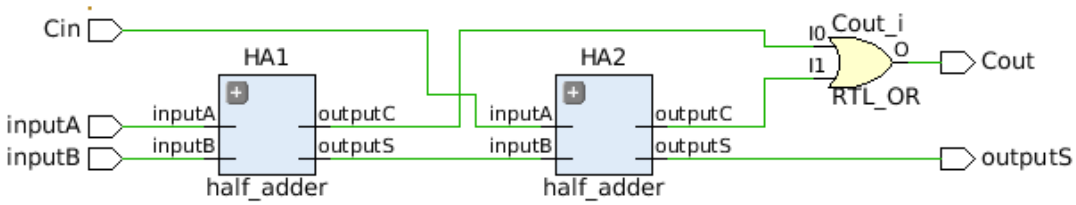
Συνδυαστικός Πλήρης Αθροιστής (Structural)

Κώδικας

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

entity full_adder_comb_struct is
    Port (inputA  : in STD_LOGIC;
          inputB  : in STD_LOGIC;
          Cin     : in STD_LOGIC;
          Cout    : out STD_LOGIC;
          outputS : out STD_LOGIC);
end full_adder_comb_struct;

architecture FA_structural of full_adder_comb_struct is

component half_adder is
    port(
     inputA  : in STD_LOGIC;
     inputB  : in STD_LOGIC;
     outputS : out STD_LOGIC;
     outputC : out STD_LOGIC
    );
end component;
    signal outS1 : STD_LOGIC;
    signal outC1 : STD_LOGIC;
    signal outC2 : STD_LOGIC;
begin
```

```
    HA1 : half_adder
     port map(
        inputA,
        inputB,
        outS1,
        outC1
     );
    HA2 : half_adder
     port map(
        Cin,
        outS1,
        outputS,
        outC2
     );
     Cout <= outC1 or outC2;

end FA_structural;
```

Διάγραμμα Δομής



Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος είναι το μονοπάτι Cin – outputS, με συνολική καθυστέρηση 4.643 ns.

Κατανάλωση πόρων

Resource	Utilization	Available	Utilization...
Slice LUTs	1	17600	0.01
IO	5	102	4.90

Ακολουθιακός Πλήρης Αθροιστής (Structural)

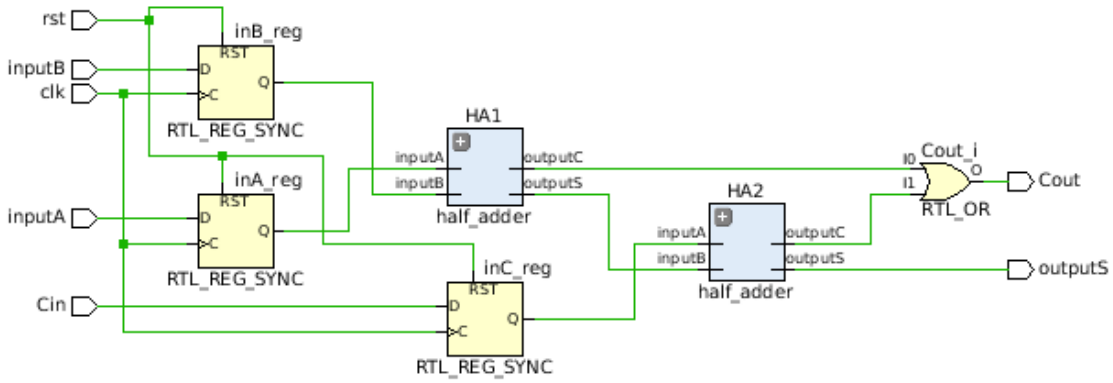
Κώδικας

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

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder_seq_struct is
    Port (clk : STD_LOGIC;
```

```
          rst : STD_LOGIC;
          inputA  : in STD_LOGIC;
          inputB  : in STD_LOGIC;
          Cin     : in STD_LOGIC;
          Cout    : out STD_LOGIC;
          outputS : out STD_LOGIC);
end full_adder_seq_struct;

architecture FA_structural of full_adder_seq_struct is

component half_adder is
    port(
     inputA  : in STD_LOGIC;
     inputB  : in STD_LOGIC;
     outputS : out STD_LOGIC;
     outputC : out STD_LOGIC
    );
end component;
    signal outS1 : STD_LOGIC;
    signal outC1 : STD_LOGIC;
    signal outC2 : STD_LOGIC;
    signal sum   : STD_LOGIC;
    signal inA   : STD_LOGIC;
    signal inB   : STD_LOGIC;
    signal inC   : STD_LOGIC;
begin
    HA1 : half_adder
     port map(
        inA,
        inB,
        outS1,
        outC1
     );
    HA2 : half_adder
     port map(
        inC,
        outS1,
        sum,
        outC2
     );
     process(clk)
     begin
     if rising_edge(clk) then
        if rst='1' then
            inA <= '0';
            inB <= '0';
            inC <= '0';
        else
            inA <= inputA;
            inB <= inputB;
            inC <= Cin;
        end if;
```

```
     end if;

     end process;
     Cout <= outC1 or outC2;
     outputS <= sum;
end FA_structural;
```

Διάγραμμα Δομής



Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος είναι το μονοπάτι inB_reg/C – outputS, με συνολική καθυστέρηση 4.166 ns.

Κατανάλωση πόρων

Resource	Utilization	Available	Utilization...
Slice LUTs	3	17600	0.02
Slice Registers	3	35200	0.01
IO	7	102	6.86
Clocking	1	32	3.12

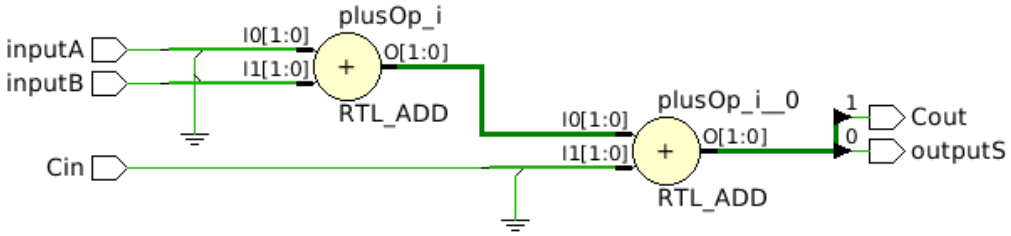
Συνδυαστικός Πλήρης Αθροιστής (Behavioral)

Κώδικας

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

entity full_adder_comb_behav is
    Port (inputA  : in STD_LOGIC;
          inputB  : in STD_LOGIC;
          Cin     : in STD_LOGIC;
          Cout    : out STD_LOGIC;
          outputS : out STD_LOGIC);
end full_adder_comb_behav;

architecture FA_behavioral of full_adder_comb_behav is
```

```
    signal sum : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal vect1 : STD_LOGIC_VECTOR(0 downto 0);
    signal vect2 : STD_LOGIC_VECTOR(0 downto 0);
    signal vect3 : STD_LOGIC_VECTOR(0 downto 0);

begin
    vect1 <= (0 => inputA);
    vect2 <= (0 => inputB);
    vect3 <= (0 => Cin);
    process(vect1, vect2, vect3)
    begin
        sum <= ('0' & vect1) + ('0'& vect2) + ('0' & vect3);

    end process;
    outputS <= sum(0);
    Cout <= sum(1);
end FA_behavioral;
```

Διάγραμμα Δομής



Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος είναι το μονοπάτι inputA – outputS, με συνολική καθυστέρηση 4.643 ns.

Κατανάλωση πόρων

Resource	Utilization	Available	Utilization...
Slice LUTs	1	17600	0.01
IO	5	102	4.90

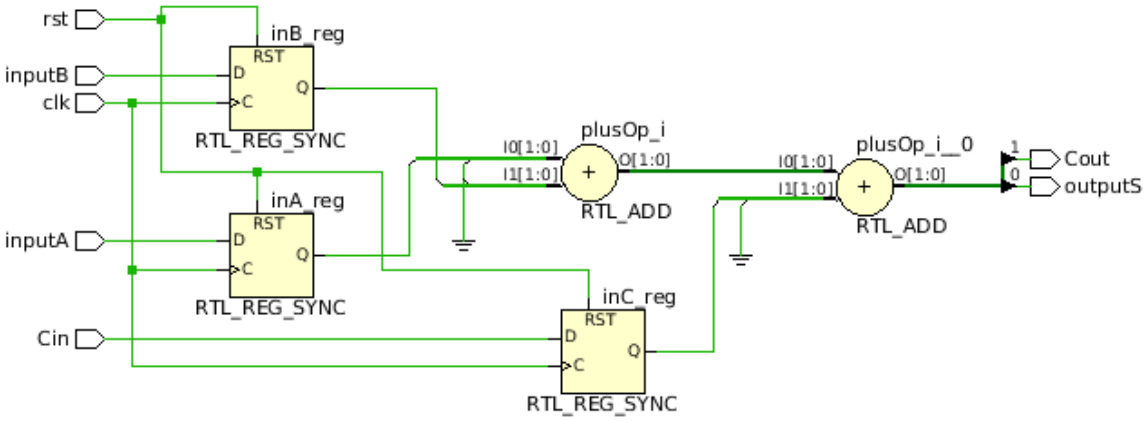
Ακολουθιακός Πλήρης Αθροιστής (Behavioral)

Κώδικας

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

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

```
entity full_adder_seq_behav is
    Port (clk : STD_LOGIC;
          rst : STD_LOGIC;
          inputA  : in STD_LOGIC;
          inputB  : in STD_LOGIC;
          Cin     : in STD_LOGIC;
          Cout    : out STD_LOGIC;
          outputS : out STD_LOGIC);
end full_adder_seq_behav;

architecture FA_behavioral of full_adder_seq_behav is

    signal sum : STD_LOGIC_VECTOR(1 downto 0);
    signal vect1 : STD_LOGIC_VECTOR(0 downto 0);
    signal vect2 : STD_LOGIC_VECTOR(0 downto 0);
    signal vect3 : STD_LOGIC_VECTOR(0 downto 0);
    signal inA   : STD_LOGIC;
    signal inB   : STD_LOGIC;
    signal inC   : STD_LOGIC;

begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if rst ='1' then
                inA <= '0';
                inB <= '0';
                inC <= '0';
            else
                inA <= inputA;
                inB <= inputB;
                inC <= Cin;
            end if;
        end if;

    end process;
    vect1 <= (0 => inA);
    vect2 <= (0 => inB);
    vect3 <= (0 => inC);
    sum <= ('0' & vect1) + ('0'& vect2) + ('0' & vect3);
    outputS <= sum(0);
    Cout <= sum(1);
end FA_behavioral;
```

Διάγραμμα Δομής



Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος είναι το μονοπάτι inB_reg/C – outputS, με συνολική καθυστέρηση 4.166 ns.

Κατανάλωση πόρων

Resource	Utilization	Available	Utilization...
Slice LUTs	3	17600	0.02
Slice Registers	3	35200	0.01
IO	7	102	6.86
Clocking	1	32	3.12

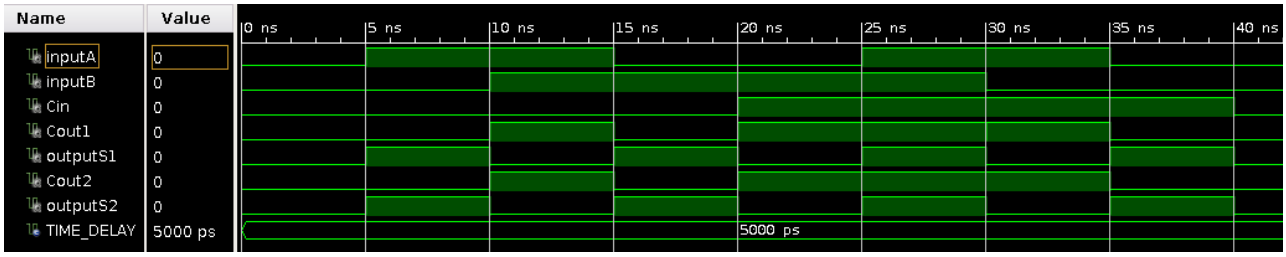
Προσομοίωση

Συνδυαστικά Κυκλώματα

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



entity full_adder_comb_tb is
end full_adder_comb_tb;

architecture Behavioral of full_adder_comb_tb is

    component full_adder_comb_struct is
        port(
         inputA  : in STD_LOGIC;
         inputB  : in STD_LOGIC;
         Cin     : in STD_LOGIC;
         Cout    : out STD_LOGIC;
         outputS : out STD_LOGIC
        );
    end component;
    component full_adder_comb_behav is
        port(
```

```
         inputA  : in STD_LOGIC;
         inputB  : in STD_LOGIC;
         Cin     : in STD_LOGIC;
         Cout    : out STD_LOGIC;
         outputS : out STD_LOGIC
        );
    end component;

    signal inputA : STD_LOGIC;
    signal inputB : STD_LOGIC;
    signal Cin : STD_LOGIC;
    signal Cout1 : STD_LOGIC;
    signal outputS1 : STD_LOGIC;
    signal Cout2 : STD_LOGIC;
    signal outputS2 : STD_LOGIC;

    constant TIME_DELAY : time := 5 ns;

begin
    uut1 : full_adder_comb_struct
     port map (
        inputA,
        inputB,
        Cin,
        Cout1,
        outputS1
     );
    uut2 : full_adder_comb_behav
     port map (
        inputA,
        inputB,
        Cin,
        Cout2,
        outputS2
     );
    stimulus : process
    begin
        inputA <= '0';
        inputB <= '0';
        Cin <= '0';
        wait for TIME_DELAY;
        inputA <= '1';
        wait for TIME_DELAY;
        inputB <= '1';
        wait for TIME_DELAY;
        inputA <= '0';
        wait for TIME_DELAY;
        Cin <= '1';
        wait for TIME_DELAY;
        inputA <= '1';
        wait for TIME_DELAY;
        inputB <= '0';
        wait for TIME_DELAY;
```

```
        inputA <= '0';
        wait for TIME_DELAY;
        Cin <= '0';
        wait;
    end process;

end Behavioral;
```

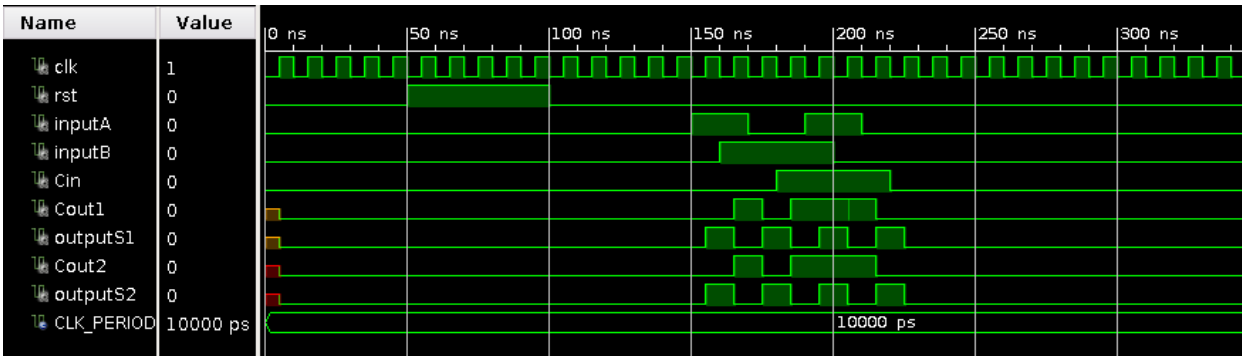


Ακολουθιακά Κυκλώματα

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


entity full_adder_seq_tb is
end full_adder_seq_tb;

architecture Behavioral of full_adder_seq_tb is

    component full_adder_seq_struct is
        port(
         clk     : STD_LOGIC;
         rst     : STD_LOGIC;
         inputA  : in STD_LOGIC;
         inputB  : in STD_LOGIC;
         Cin     : in STD_LOGIC;
         Cout    : out STD_LOGIC;
         outputS : out STD_LOGIC
        );
    end component;
    component full_adder_seq_behav is
        port(
         clk     : STD_LOGIC;
         rst     : STD_LOGIC;
         inputA  : in STD_LOGIC;
         inputB  : in STD_LOGIC;
         Cin     : in STD_LOGIC;
         Cout    : out STD_LOGIC;
         outputS : out STD_LOGIC
        );
    end component;

    signal clk : STD_LOGIC;
    signal rst : STD_LOGIC;
```

```
    signal inputA : STD_LOGIC;
    signal inputB : STD_LOGIC;
    signal Cin : STD_LOGIC;
    signal Cout1 : STD_LOGIC;
    signal outputS1 : STD_LOGIC;
    signal Cout2 : STD_LOGIC;
    signal outputS2 : STD_LOGIC;

    constant CLK_PERIOD : time := 10 ns;

begin
    uut1 : full_adder_seq_struct
     port map (
        clk,
        rst,
        inputA,
        inputB,
        Cin,
        Cout1,
        outputS1
     );
    uut2 : full_adder_seq_behav
     port map (
        clk,
        rst,
        inputA,
        inputB,
        Cin,
        Cout2,
        outputS2
     );
    stimulus : process
    begin
        rst <= '0';
        inputA <= '0';
        inputB <= '0';
        Cin <= '0';
        wait for CLK_PERIOD*5;
        rst <= '1';
        wait for CLK_PERIOD*5;
        rst <= '0';
        wait for CLK_PERIOD*5;
        inputA <= '1';
        wait for CLK_PERIOD;
        inputB <= '1';
        wait for CLK_PERIOD;
        inputA <= '0';
        wait for CLK_PERIOD;
        Cin <= '1';
        wait for CLK_PERIOD;
        inputA <= '1';
        wait for CLK_PERIOD;
        inputB <= '0';
```

```
        wait for CLK_PERIOD;
        inputA <= '0';
        wait for CLK_PERIOD;
        Cin <= '0';
        wait;
    end process;

  generate_clock : process
  begin
    clk <= '0';
    wait for CLK_PERIOD/2;
    clk <= '1';
    wait for CLK_PERIOD/2;
  end process;

end Behavioral;
```



Ζήτημα 3: 4-bit Αθροιστής Διάδοσης Κρατουμένου (RCA)

Υλοποιήθηκαν με περιγραφή δομής συνδυαστικός RCA χρησιμοποιώντας τον συνδυαστικό πλήρη αθροιστή του ερωτήματος 2.a και ακολουθιακός RCA χρησιμοποιώντας τον ακολουθιακό πλήρη αθροιστή του ερωτήματος 2.b.

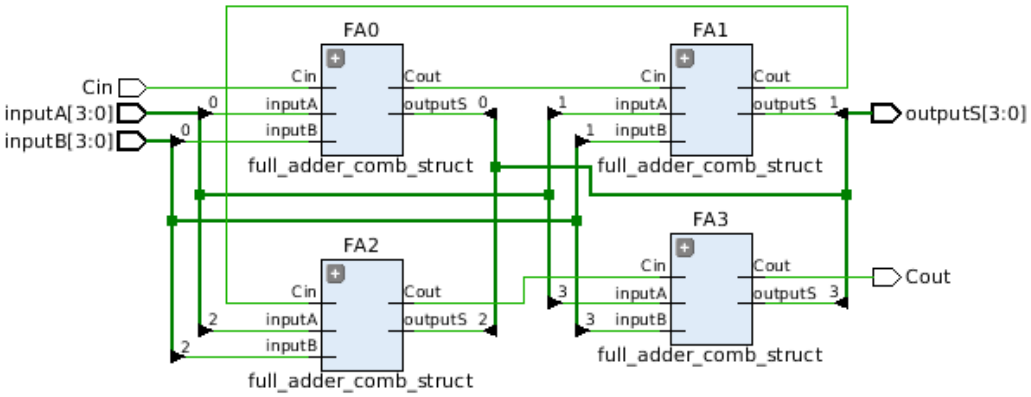
4-bit Συνδυαστικός Αθροιστής

Κώδικας

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

entity adder4_comb is
    Port (
     inputA  : in STD_LOGIC_VECTOR(3 downto 0);
     inputB  : in STD_LOGIC_VECTOR(3 downto 0);
     Cin     : in STD_LOGIC;
     outputS : out STD_LOGIC_VECTOR(3 downto 0);
     Cout    : out STD_LOGIC
     );
end adder4_comb;

architecture Structural of adder4_comb is
```

```
component full_adder_comb_struct is
    port(
     inputA  : in STD_LOGIC;
     inputB  : in STD_LOGIC;
     Cin     : in STD_LOGIC;
     Cout    : out STD_LOGIC;
     outputS : out STD_LOGIC
    );
end component;

    signal Carry : STD_LOGIC_VECTOR(2 downto 0);

begin
    FA0 : full_adder_comb_struct
     port map(
        inputA(0),
        inputB(0),
        Cin,
        Carry(0),
        outputS(0)
    );
    FA1 : full_adder_comb_struct
     port map(
        inputA(1),
        inputB(1),
        Carry(0),
        Carry(1),
        outputS(1)
        );
    FA2 : full_adder_comb_struct
     port map(
        inputA(2),
        inputB(2),
        Carry(1),
        Carry(2),
        outputS(2)
        );
    FA3 : full_adder_comb_struct
     port map(
        inputA(3),
        inputB(3),
        Carry(2),
        Cout,
        outputS(3)
        );

end Structural;
```

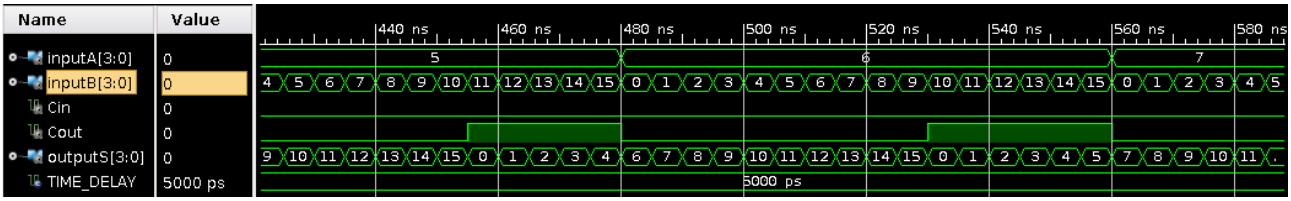
Διάγραμμα Δομής



Προσομοίωση

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


entity adder4_comb_tb is
end adder4_comb_tb;

architecture Behavioral of adder4_comb_tb is

    component adder4_comb is
        port(
         inputA  : in STD_LOGIC_VECTOR(3 DOWNTO 0);
         inputB  : in STD_LOGIC_VECTOR(3 DOWNTO 0);
         Cin     : in STD_LOGIC;
         outputS : out STD_LOGIC_VECTOR(3 DOWNTO 0);
         Cout    : out STD_LOGIC
        );
    end component;

    signal inputA : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal inputB : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal Cin : STD_LOGIC;
    signal Cout : STD_LOGIC;
    signal outputS : STD_LOGIC_VECTOR(3 DOWNTO 0);

    constant TIME_DELAY : time := 5 ns;

begin

    uut : adder4_comb
     port map(
        inputA,
        inputB,
        Cin,
        outputS,
        Cout
```

```
     );

    stimulus : process
    begin
        Cin <= '0';
        for i in 0 to 15 loop
            inputA <= std_logic_vector(to_unsigned(i, 4));
            for j in 0 to 15 loop
                inputB <= std_logic_vector(to_unsigned(j, 4));
                wait for TIME_DELAY;
            end loop;
        end loop;
        Cin <= '1';
        for i in 0 to 15 loop
             inputA <= std_logic_vector(to_unsigned(i, 4));
             for j in 0 to 15 loop
                 inputB <= std_logic_vector(to_unsigned(j, 4));
                 wait for TIME_DELAY;
             end loop;
         end loop;
         inputA <= (others => '0');
         inputB <= (others => '0');
         Cin <= '0';
         wait;
    end process;

end Behavioral;
```



Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος είναι το μονοπάτι inputA[0] – outputS[3], με συνολική καθυστέρηση 5.497 ns.

Κατανάλωση Πόρων

Resource	Utilization	Available	Utilization...
Slice LUTs	4	17600	0.02
IO	14	102	13.73

4-bit Ακολουθιακός Αθροιστής

Κώδικας

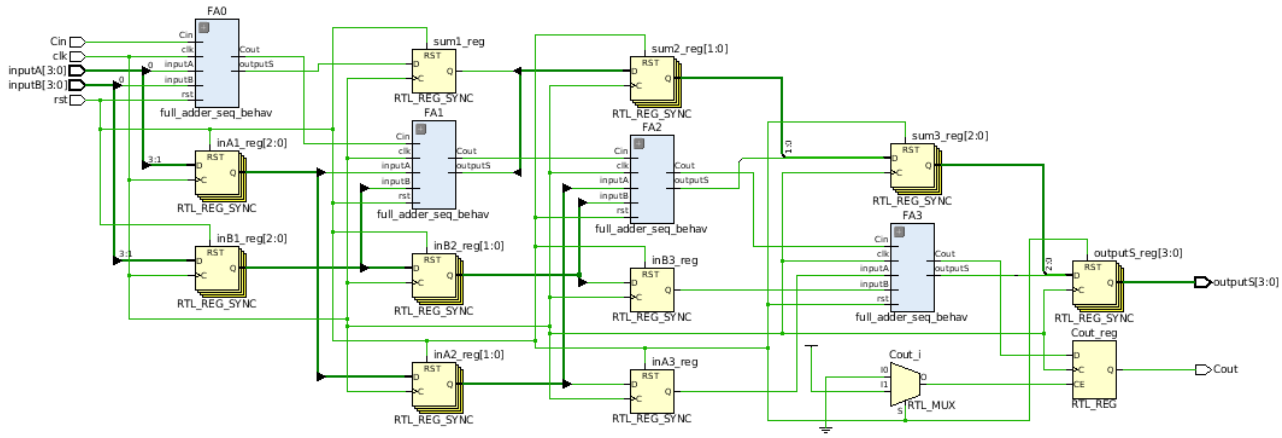
```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity adder4_seq is
    Port (
     clk     : STD_LOGIC;
     rst     : STD_LOGIC;
     inputA  : in STD_LOGIC_VECTOR(3 downto 0);
     inputB  : in STD_LOGIC_VECTOR(3 downto 0);
     Cin     : in STD_LOGIC;
     outputS : out STD_LOGIC_VECTOR(3 downto 0);
     Cout    : out STD_LOGIC
     );
end adder4_seq;



architecture Behavioral of adder4_seq is
component full_adder_seq_behav is
    port(
     clk     : STD_LOGIC;
     rst     : STD_LOGIC;
     inputA  : in STD_LOGIC;
     inputB  : in STD_LOGIC;
     Cin     : in STD_LOGIC;
     Cout    : out STD_LOGIC;
     outputS : out STD_LOGIC
    );
end component;

    signal inA1 : STD_LOGIC_VECTOR(2 downto 0);
    signal inB1 : STD_LOGIC_VECTOR(2 downto 0);
    signal inA2 : STD_LOGIC_VECTOR(1 downto 0);
    signal inB2 : STD_LOGIC_VECTOR(1 downto 0);
    signal inA3 : STD_LOGIC;
    signal inB3 : STD_LOGIC;

    signal Carry : STD_LOGIC_VECTOR(3 downto 0);
    signal Sout : STD_LOGIC_VECTOR(3 downto 0);
    signal sum1 : STD_LOGIC;
    signal sum2 : STD_LOGIC_VECTOR(1 downto 0);
    signal sum3 : STD_LOGIC_VECTOR(2 downto 0);
begin
    FA0 : full_adder_seq_behav
 port map(
    clk,
    rst,
    inputA(0),
    inputB(0),
    Cin,
    Carry(0),
    Sout(0)
);
FA1 : full_adder_seq_behav
 port map(
    clk,
```

```
    rst,
    inA1(0),
    inB1(0),
    Carry(0),
    Carry(1),
    Sout(1)
    );
FA2 : full_adder_seq_behav
 port map(
    clk,
    rst,
    inA2(0),
    inB2(0),
    Carry(1),
    Carry(2),
    Sout(2)
    );
FA3 : full_adder_seq_behav
 port map(
    clk,
    rst,
    inA3,
    inB3,
    Carry(2),
    Carry(3),
    Sout(3)
    );

    process(clk)
    begin
        if clk'event and clk='1' then
            if rst='1' then
                inA1 <= (others => '0');
                inB1 <= (others => '0');
                inA2 <= (others => '0');
                inB2 <= (others => '0');
                inA3 <= '0';
                inB3 <= '0';
                outputS <= (others => '0');
                sum1 <= '0';
                sum2 <= (others => '0');
                sum3 <= (others => '0');
            else
                inA3 <= inA2(1);
                inA2 <= (1 => inA1(2), 0 => inA1(1));
                inA1 <= (2 => inputA(3), 1 => inputA(2), 0 =>
inputA(1));
                inB3 <= inB2(1);
                inB2 <= (1 => inB1(2), 0 => inB1(1));
                inB1 <= (2 => inputB(3), 1 => inputB(2), 0 =>
inputB(1));
                outputS <= Sout(3) & sum3;
                sum3 <= Sout(2) & sum2;
```

```
                sum2 <= (1 => Sout(1), 0 => sum1);
                sum1 <= Sout(0);
                Cout <= Carry(3);
            end if;
        end if;
    end process;


end Behavioral;
```

Διάγραμμα Δομής

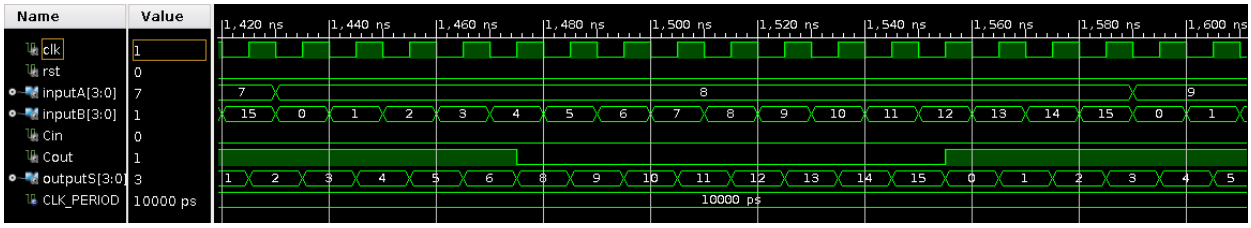


Προσομοίωση

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


entity adder4_seq_tb is
end adder4_seq_tb;

architecture Behavioral of adder4_seq_tb is

    component adder4_seq is
        port(
         clk     : STD_LOGIC;
         rst     : STD_LOGIC;
         inputA  : in STD_LOGIC_VECTOR(3 DOWNTO 0);
         inputB  : in STD_LOGIC_VECTOR(3 DOWNTO 0);
         Cin     : in STD_LOGIC;
         outputS : out STD_LOGIC_VECTOR(3 DOWNTO 0);
         Cout    : out STD_LOGIC
        );
    end component;

    signal clk : STD_LOGIC;
    signal rst : STD_LOGIC;
```

```
    signal inputA : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal inputB : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal Cin : STD_LOGIC;
    signal Cout : STD_LOGIC;
    signal outputS : STD_LOGIC_VECTOR(3 DOWNTO 0);

    constant CLK_PERIOD : time := 10 ns;

begin

    uut : adder4_seq
     port map(
        clk,
        rst,
        inputA,
        inputB,
        Cin,
        outputS,
        Cout
     );

    stimulus : process
    begin
        rst <= '0';
        inputA <= (others => '0');
        inputB <= (others => '0');
        Cin <= '0';
        wait for CLK_PERIOD*5;
        rst <= '1';
        wait for CLK_PERIOD*5;
        rst <= '0';
        wait for CLK_PERIOD*5;
        for i in 0 to 15 loop
            inputA <= std_logic_vector(to_unsigned(i, 4));
            for j in 0 to 15 loop
                inputB <= std_logic_vector(to_unsigned(j, 4));
                wait for CLK_PERIOD;
            end loop;
        end loop;
        Cin <= '1';
        for i in 0 to 15 loop
             inputA <= std_logic_vector(to_unsigned(i, 4));
             for j in 0 to 15 loop
                 inputB <= std_logic_vector(to_unsigned(j, 4));
                 wait for CLK_PERIOD;
             end loop;
         end loop;
         inputA <= (others => '0');
         inputB <= (others => '0');
         Cin <= '0';
         wait;
    end process;
```

```
  generate_clock : process
  begin
    clk <= '0';
    wait for CLK_PERIOD/2;
    clk <= '1';
    wait for CLK_PERIOD/2;
  end process;

end Behavioral;
```



Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος (ή ένα εξ αυτών με ίση καθυστέρηση) είναι το μονοπάτι Cout_reg/C – Cout, με συνολική καθυστέρηση 3.441 ns.

Κατανάλωση Πόρων

Resource	Utilization	Available	Utilization...
Slice LUTs	9	17600	0.05
Slice Registers	32	35200	0.09
IO	16	102	15.69
Clocking	1	32	3.12

Ζήτημα 4

Με είσοδο δύο *BCD ψηφία* και *κρατούμενο εισόδου* αναμένουμε στην έξοδο το σωστό *BCD ψηφίο* και το δέον *κρατούμενο εξόδου* (που προκύπτουν από την άθροιση των δύο ψηφίων εισόδου και του κρατουμένου εισόδου). Ο έλεγχος της σωστής αναπαράστασης επιτυγχάνεται μέσω του *module* **BCD_check** το οποίο δέχεται ως είσοδο το αποτέλεσμα του αθροίσματος και το κρατούμενο εξόδου και ελέγχει εάν το άρθοισμα είναι μεγαλύτερο του εννέα ή το κρατούμενο εξόδου είναι *set* οπότε προσθέτει το έξι στο άθροισμα.

(κώδικας BCD_check + βοηθητικών modules)

(or2)

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

entity or2 is
  port(
    a : in STD_LOGIC;
    b : in STD_LOGIC;
    c : out STD_LOGIC
  );
end or2;

architecture structural of or2 is
  begin
    c <= a or b;
end structural;
```

(and2)

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

entity and2 is
  port(
    a : in STD_LOGIC;
    b : in STD_LOGIC;
    c : out STD_LOGIC
  );
end and2;

architecture structural of and2 is
  begin
    c <= a and b;
end structural;
```

(BCD_check)

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

entity BCD_check is
  port(
```

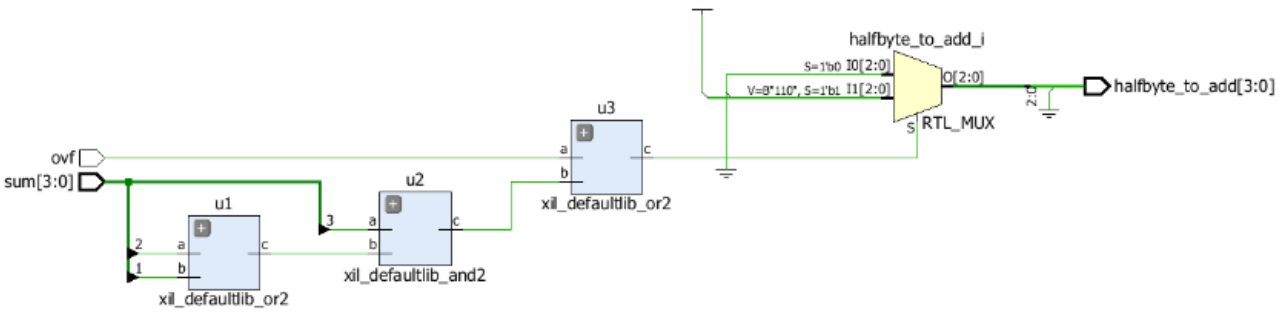
```
    ovf : in STD_LOGIC;
    sum : in STD_LOGIC_VECTOR(3 downto 0);
    halfbyte_to_add : out STD_LOGIC_VECTOR(3 downto 0)
  );
end BCD_check;

architecture structural of BCD_check is

  component or2 is
    port (
      a : in STD_LOGIC;
      b : in STD_LOGIC;
      c : out STD_LOGIC
    );
  end component;

  component and2 is
    port (
      a : in STD_LOGIC;
      b : in STD_LOGIC;
      c : out STD_LOGIC
    );
  end component;

  signal a1 : STD_LOGIC;
  signal a2 : STD_LOGIC;
  signal a3 : STD_LOGIC;

    begin

     u1 : or2
     port map (
       a => sum(2),
       b => sum(1),
       c => a1
     );

     u2 : and2
     port map (
       a => sum(3),
       b => a1,
       c => a2
     );

     u3 : or2
     port map (
       a => ovf,
       b => a2,
       c => a3
     );

     with a3 select halfbyte_to_add <=
       "0000" when '0',
       "0110" when '1';

end structural;
```

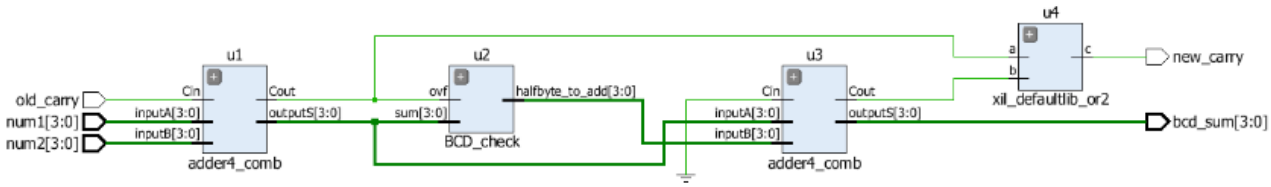
(κώδικας πλήρους αθροιστή BCD)

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

entity BCD_FA is
  port(
    num1 : in STD_LOGIC_VECTOR(3 downto 0);
    num2 : in STD_LOGIC_VECTOR(3 downto 0);
    old_carry : in STD_LOGIC;
    bcd_sum : out STD_LOGIC_VECTOR(3 downto 0);
    new_carry : out STD_LOGIC
  );
end BCD_FA;


architecture structural of BCD_FA is

  component or2 is
    port(
     a : in STD_LOGIC;
     b : in STD_LOGIC;
     c : out STD_LOGIC
    );
   end component;

  component adder4_comb is
    port(
      inputA : in STD_LOGIC_VECTOR(3 downto 0);
      inputB : in STD_LOGIC_VECTOR(3 downto 0);
      Cin : in STD_LOGIC;
      outputS : out STD_LOGIC_VECTOR(3 downto 0);
      Cout : out STD_LOGIC
    );
  end component;

  component BCD_check is
    port(
      ovf : in STD_LOGIC;
      sum : in STD_LOGIC_VECTOR(3 downto 0);
      halfbyte_to_add : out STD_LOGIC_VECTOR(3 downto 0)
    );
   end component;

  signal temp_bcd_sum : STD_LOGIC_VECTOR(3 downto 0);
  signal temp_new_carry1 : STD_LOGIC;
  signal temp_new_carry2 : STD_LOGIC;
  signal zero_six : STD_LOGIC_VECTOR(3 downto 0);

  begin

    u1 : adder4_comb
    port map (
      inputA => num1,
      inputB => num2,
      Cin => old_carry,
      outputS => temp_bcd_sum,
      Cout => temp_new_carry1
```

```
    );

    u2 : BCD_check
    port map (
      ovf => temp_new_carry1,
      sum => temp_bcd_sum,
      halfbyte_to_add => zero_six
    );

    u3 : adder4_comb
    port map (
      inputA => temp_bcd_sum,
      inputB => zero_six,
      Cin => '0',
      outputS => bcd_sum,
      Cout => temp_new_carry2
    );

    u4 : or2
    port map (
      a => temp_new_carry1,
      b => temp_new_carry2,
      c => new_carry
    );

end structural;
```

(RTL Schematic BCD_check/BCD_FA)

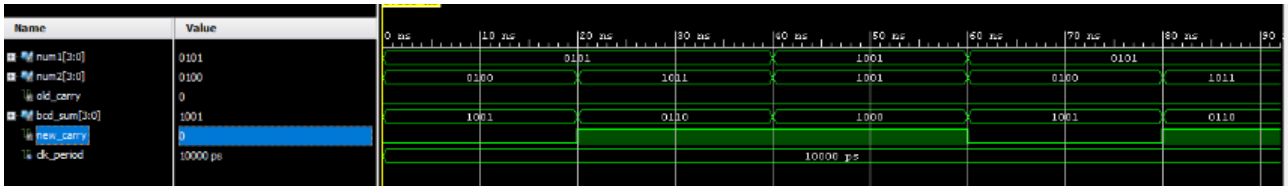




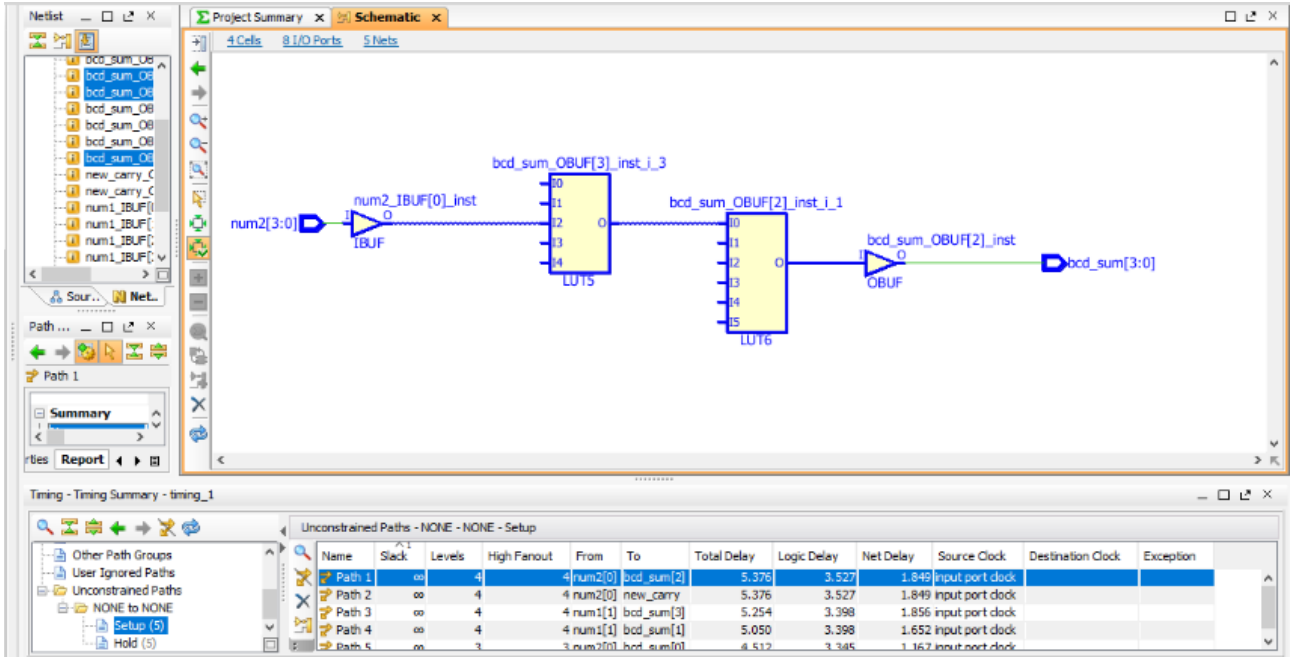
(testbench για τον πλήρη BCD αθροιστή)
(κώδικας testbench)

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

entity BCD_FA_tb is
end BCD_FA_tb;

architecture testbench of BCD_FA_tb is

  component BCD_FA is
    port(
    num1 : in STD_LOGIC_VECTOR(3 downto 0);
    num2 : in STD_LOGIC_VECTOR(3 downto 0);
    old_carry : in STD_LOGIC;
    bcd_sum : out STD_LOGIC_VECTOR(3 downto 0);
    new_carry : out STD_LOGIC
    );
  end component;

  signal num1 : STD_LOGIC_VECTOR(3 downto 0);
  signal num2 : STD_LOGIC_VECTOR(3 downto 0);
  signal old_carry : STD_LOGIC;
  signal bcd_sum : STD_LOGIC_VECTOR(3 downto 0);
  signal new_carry : STD_LOGIC;

  constant wait_period : time := 10 ns ;

  begin
    uut : BCD_FA
    port map (
      num1 => num1,
      num2 => num2,
      old_carry => old_carry,
      bcd_sum => bcd_sum,
      new_carry => new_carry
    );
```

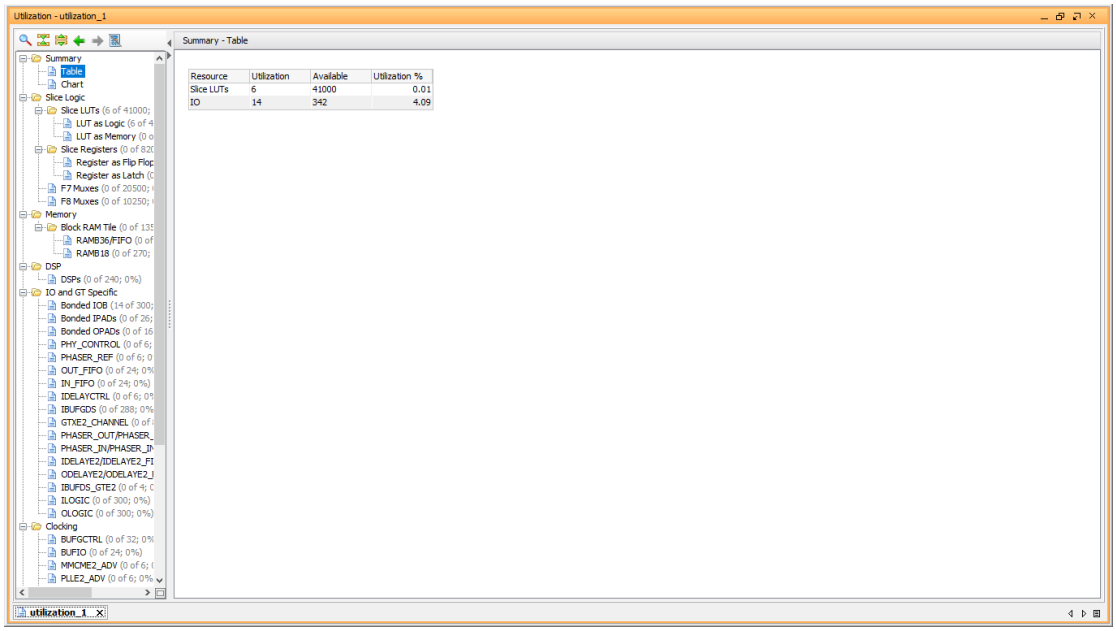
```
    stimulus : process
      begin
        num1 <= "0101";
        num2 <= "0100";
        old_carry <= '0';
        wait for 2*wait_period;

        num2 <= "1011";
        wait for 2*wait_period;

        num1 <= "1001";
        num2 <= "1001";
        wait for 2*wait_period;
    end process;

end testbench;
```



(Timing Summary)



(Utilization)



Ζήτημα 5

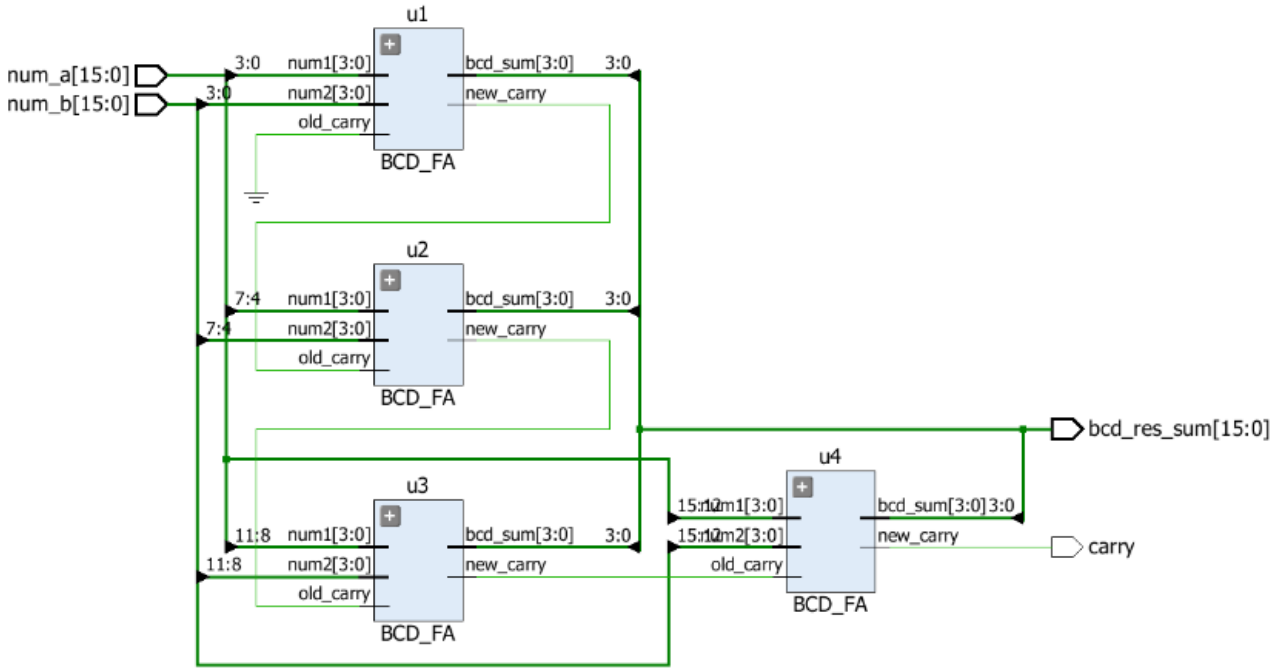
Βάσει του πλήρους αθροιστή του Ζητήματος 4 υλοποιείται ο παράλληλος BCD αθροιστής 4 BCD ψηφίων. Ο **BCD4_PA** δέχεται ως είσοδος δύο 4-ψήφιους BCD αριθμούς και κρατούμενο εισόδου και υπολογίζει το αποτέλεσμα του αθροίσματος των δύο αυτών αριθμών καθώς και το κρατούμενο εξόδου.

(κώδικας παράλληλου BCD αθροιστή 4 ψηφίων)

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

entity BCD4_PA is
  port (
    num_a : in STD_LOGIC_VECTOR(15 downto 0);
    num_b : in STD_LOGIC_VECTOR(15 downto 0);
    bcd_res_sum : out STD_LOGIC_VECTOR(15 downto 0);
    carry : out STD_LOGIC
  );
end BCD4_PA;

architecture structural of BCD4_PA is

  component BCD_FA is
    port(
      num1 : in STD_LOGIC_VECTOR(3 downto 0);
      num2 : in STD_LOGIC_VECTOR(3 downto 0);
      old_carry : in STD_LOGIC;
      bcd_sum : out STD_LOGIC_VECTOR(3 downto 0);
      new_carry : out STD_LOGIC
    );
  end component;

  signal temp1 : STD_LOGIC;
  signal temp2 : STD_LOGIC;
```

```
  signal temp3 : STD_LOGIC;
  signal temp4 : STD_LOGIC;

  begin
    temp1 <= '0';

    u1 : BCD_FA
    port map (
      num1 => num_a(3 downto 0),
      num2 => num_b(3 downto 0),
      old_carry => temp1,
      bcd_sum => bcd_res_sum(3 downto 0),
      new_carry => temp2
    );

    u2 : BCD_FA
    port map (
      num1 => num_a(7 downto 4),
      num2 => num_b(7 downto 4),
      old_carry => temp2,
      bcd_sum => bcd_res_sum(7 downto 4),
      new_carry => temp3
    );

    u3 : BCD_FA
    port map (
      num1 => num_a(11 downto 8),
      num2 => num_b(11 downto 8),
      old_carry => temp3,
      bcd_sum => bcd_res_sum(11 downto 8),
      new_carry => temp4
    );

    u4 : BCD_FA
    port map (
      num1 => num_a(15 downto 12),
      num2 => num_b(15 downto 12),
      old_carry => temp4,
      bcd_sum => bcd_res_sum(15 downto 12),
      new_carry => carry
    );

end structural;
```

(RTL Schematic)

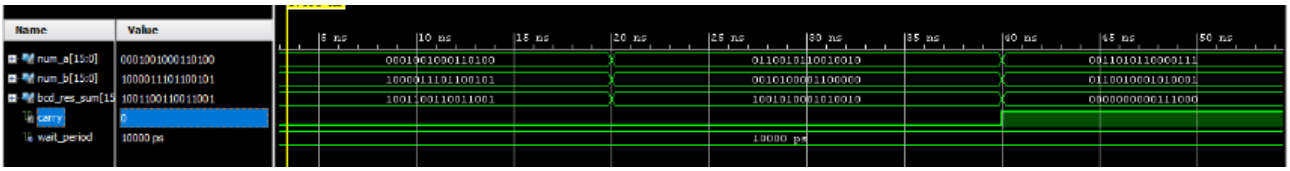


(testbench για τον παράλληλο BCD αθροιστή 4 ψηφίων)

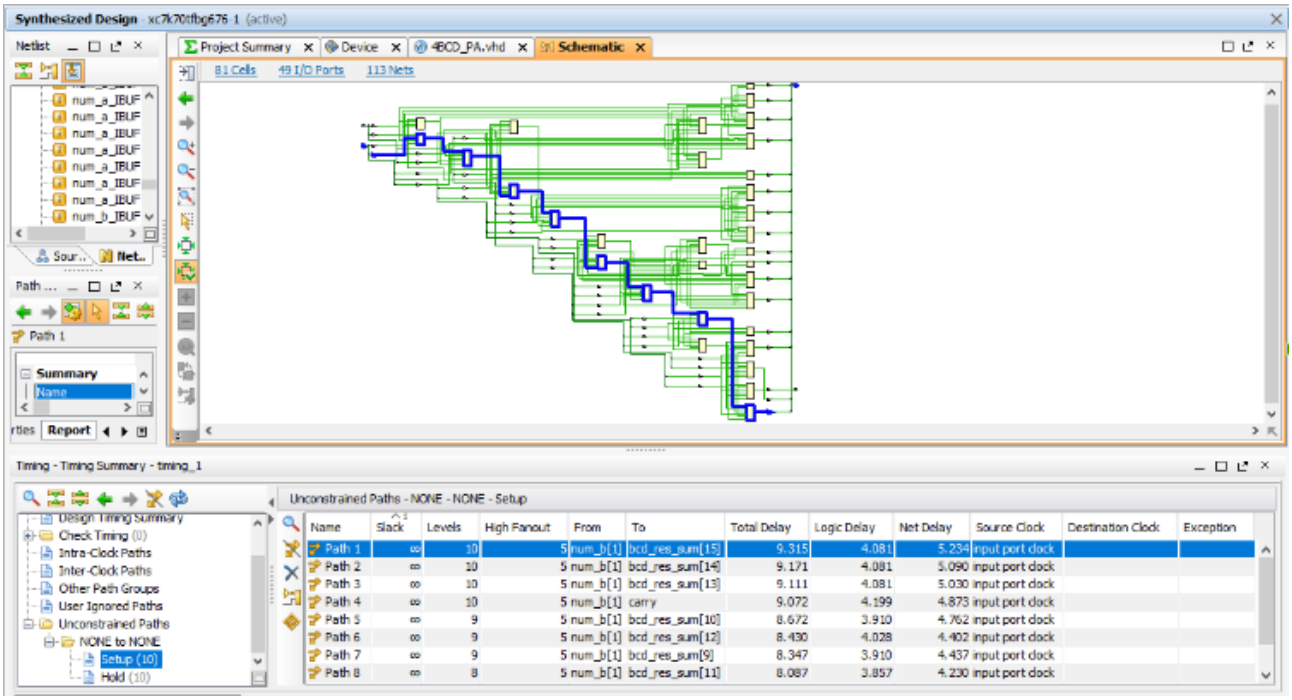
(κώδικας testbench)

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

entity BCD4_PA_tb is
end BCD4_PA_tb;

architecture testbench of BCD4_PA_tb is

  component BCD4_PA is
    port (
    num_a : in STD_LOGIC_VECTOR(15 downto 0);
    num_b : in STD_LOGIC_VECTOR(15 downto 0);
    bcd_res_sum : out STD_LOGIC_VECTOR(15 downto 0);
    carry : out STD_LOGIC
    );
  end component;

  signal num_a : STD_LOGIC_VECTOR(15 downto 0);
  signal num_b : STD_LOGIC_VECTOR(15 downto 0);
  signal bcd_res_sum : STD_LOGIC_VECTOR(15 downto 0);
  signal carry : STD_LOGIC;

  constant wait_period : time := 10ns;

  begin
    uut : BCD4_PA
    port map (
      num_a => num_a,
```

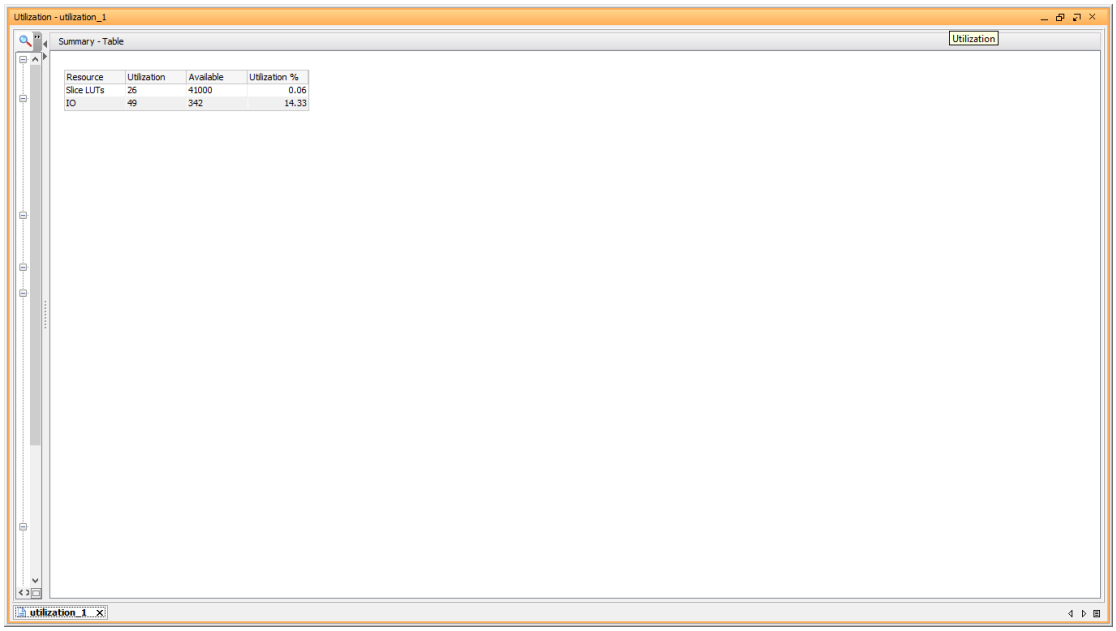
```
      num_b => num_b,
      bcd_res_sum => bcd_res_sum,
      carry => carry
    );

    stimulus : process
      begin

        num_a <= "0001001000110100"; -- 1234
        num_b <= "1000011101100101"; -- 8765
        wait for 2*wait_period;

        num_a <= "0110010110010010"; -- 6592
        num_b <= "0010100001100000"; -- 2860
        wait for 2*wait_period;

        num_a <= "0011010110000111"; -- 3587
        num_b <= "0110010001010001"; -- 6451
        wait for 2*wait_period;
    end process;


end testbench;
```



(Timing Summary)



(**Utilization**)



Ζήτημα 6: Συστολικός Πολλαπλασιαστής

Ακολουθεί η υλοποίηση του συστολικού πολλαπλασιαστή, η οποία όμως δε λειτουργεί ορθά. Επιπλέον ο πλήρης αθροιστής που χρησιμοποιείται είναι μια παραλλαγή του full_adder_seq_behav του ερωτήματος 2.a στον οποίο ο καταχωρητής είναι στην έξοδο και όχι την αρχή του κυκλώματος και παρατίθεται ο κώδικάς του μαζί με του πολλαπλασιαστή.

Κώδικας

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



entity full_adder_seq_behav is
    Port (clk    : in STD_LOGIC;
          rst    : in STD_LOGIC;
          inputA : in STD_LOGIC;
          inputB : in STD_LOGIC;
          Cin    : in STD_LOGIC;
          Cout   : out STD_LOGIC;
          outputS : out STD_LOGIC);
end full_adder_seq_behav;

architecture FA_behavioral of full_adder_seq_behav is

    signal sum : STD_LOGIC_VECTOR(1 downto 0);
```

```
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if rst ='1' then

                sum <= (others => '0');
            else
                sum <= ('0' & inputA) + ('0'& inputB) + ('0' &
Cin);
            end if;
        end if;

    end process;

    outputS <= sum(0);
    Cout <= sum(1);
end FA_behavioral;

entity mul_cell is
   port
   (
      ai : in std_logic;
      bi : in std_logic;
      si : in std_logic;
      ci : in std_logic;
      clk : in std_logic;
      rst : in std_logic;
      ao : out std_logic;
      bo : out std_logic;
      so : out std_logic;
      co : out std_logic
   );
end mul_cell;

architecture structural of mul_cell is

component full_adder_seq_behav is
    Port (clk    : in STD_LOGIC;
          rst     : in STD_LOGIC;
          inputA  : in STD_LOGIC;
          inputB  : in STD_LOGIC;
          Cin     : in STD_LOGIC;
          Cout    : out STD_LOGIC;
          outputS : out STD_LOGIC);
end component;



signal ab  :   std_logic;
signal s   :    std_logic;
```

```
signal s1  :    std_logic;
signal a   :    std_logic;
signal b   :    std_logic;
signal c   :    std_logic;
signal b1  :    std_logic;
signal a1  :    std_logic;
signal a2  :    std_logic;


begin

   FA : full_adder_seq_behav
   PORT MAP (clk =>clk,
             rst =>rst,
             inputA => s,
           inputB =>ab,
           Cin => c,
           Cout => co,
           outputS =>so);


   process(clk)
    begin
        if clk'event and clk='1' then
         if rst ='1' then

            ab <= '0';
            s   <= '0';

            a  <= '0';
            a1 <= '0';
            a2 <= '0';
            b  <= '0';
            b1 <= '0';
            c  <= '0';
         else
             ab <= ai and bi;
             s  <= si;
            a <= ai;
            a1 <= a;
            a2 <= a1;
            b <= bi;
            b1 <= b;
            c <= ci;

         end if;
      end if;

   end process;
   ao <= a2;
   bo <= b1;

end structural;
```

```
entity multiplier is
   port
   (
        clk,rst : in std_logic;
            a : in std_logic_vector(3 downto 0);
            b : in std_logic_vector(3 downto 0);
            p : out std_logic_vector(7 downto 0)

   );
end multiplier;

architecture structural of multiplier is

subtype a_word is std_logic_vector(3 downto 0);
type a_word_array is array(natural range <>) of a_word;
signal ai,ao,bi,bo,si,so,ci,co : a_word_array(3 downto 0);

signal cout_edge : std_logic_vector(2 downto 0);
signal cout_edge1 : std_logic_vector(2 downto 0);
signal cout_edge2 : std_logic_vector(2 downto 0);

signal a1 : std_logic_vector(1 downto 0;
signal a2 : std_logic_vector(2 downto 0);
signal a3 : std_logic_vector(3 downto 0);

signal b1 : std_logic_vector(1 downto 0);
signal b2 : std_logic_vector(4 downto 0);
signal b3 : std_logic_vector(6 downto 0);

signal p0 : std_logic_vector(10 downto 0);
signal p1 : std_logic_vector(7 downto 0);
signal p2 : std_logic_vector(5 downto 0);
signal p3 : std_logic_vector(3 downto 0);
signal p4 : std_logic_vector(2 downto 0);
signal p5 : std_logic_vector(1 downto 0);


component mul_cell is
   port
      (
         ai : in std_logic;
         bi : in std_logic;
         si : in std_logic;
         ci : in std_logic;
         clk : in std_logic;
         rst : in std_logic;
         ao : out std_logic;
         bo : out std_logic;
         so : out std_logic;
         co : out std_logic
      );
end component;
```

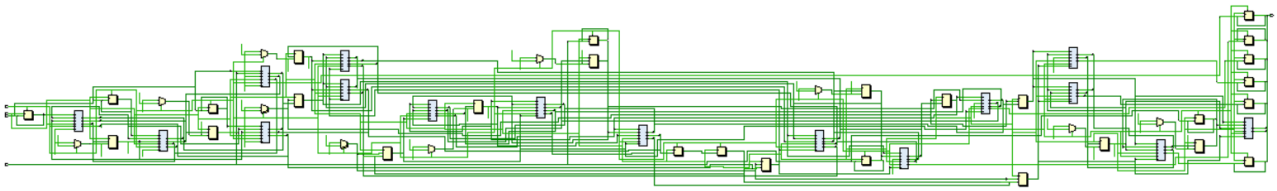
```
begin
   -- cell generation
   gcb: for i in 0 to 3 generate
      gca: for j in 0 to 3 generate
         gc: mul_cell port map
            (
               ai => ai(i)(j),
               bi => bi(i)(j),
               si => si(i)(j),
               ci => ci(i)(j),
               clk => clk,
               rst => rst,
               ao => ao(i)(j),
               bo => bo(i)(j),
               so => so(i)(j),
               co => co(i)(j)
            );
      end generate;
   end generate;

-- intermediate wires generation
gasw: for i in 1 to 3 generate
         ai(i) <= ao(i-1);
         si(i) <= cout_edge(i-1) & so(i-1)(3 downto 1);  --correct
for couts in edges
      end generate;


gbciw: for i in 0 to 3 generate
   gbcjw: for j in 1 to 3 generate
            bi(i)(j) <= bo(i)(j-1);
            ci(i)(j) <= co(i)(j-1);
         end generate;
      end generate;

-- input connections(input zeros )
gsi:  si(0) <= (others => '0');

gci:  for i in 0 to 3 generate
            ci(i)(0) <= '0';
         end generate;

process(clk)
   begin
      if clk'event and clk='1' then
         if rst='1' then
             cout_edge <= (others => '0');
             cout_edge1 <= (others => '0');
             cout_edge2 <= (others => '0');
            a1 <= (others => '0');
            a2 <= (others => '0');
            a3 <= (others => '0');
            b1 <= (others => '0');
```

```
            b2 <= (others => '0');
            b3 <= (others => '0');
            p0 <= (others => '0');
            p1 <= (others => '0');
            p2 <= (others => '0');
            p3 <= (others => '0');
            p4 <= (others => '0');
            p5 <= (others => '0');

         else

             cout_edge2 <= (2 => co(2)(3) , 1 => co (1)(3), 0 =>
co(0)(3));
             cout_edge1 <= cout_edge2;
             cout_edge <= cout_edge1;

            -- additional flipflops at b inputs
            b1(0) <= b(1);
            b1(1) <= b1(0);
            b1(2) <= b1(1);

            b2(0) <= b(2);
            b2(1) <= b2(0);
            b2(2) <= b2(1);
            b2(3) <= b2(2);
            b2(4) <= b2(3);

            b3(0) <= b(3);
            b3(1) <= b3(0);
            b3(2) <= b3(1);
            b3(3) <= b3(2);
            b3(4) <= b3(3);
            b3(5) <= b3(4);
            b3(6) <= b3(5);

            -- additional flipflops at a inputs
              a1 <= a(1);
              --a1(1) <= a1(0);

              a2(0) <= a(2);
              a2(1) <= a2(0);
              a2(2) <= a2(1);

              a3(0) <= a(3);
              a3(1) <= a3(0);
              a3(2) <= a3(1);
              a3(3) <= a3(2);

            -- additional flip flops at outputs
            p0(0) <= so(0)(0);
            p0(1) <= p0(0);
            p0(2) <= p0(1);
            p0(3) <= p0(2);
```

```
            p0(4) <= p0(3);
            p0(5) <= p0(4);
            p0(6) <= p0(5);
            p0(7) <= p0(6);
            p0(8) <= p0(7);
            p0(9) <= p0(8);
            p0(10) <= p0(9);

            p1(0) <= so(1)(0);
            p1(1) <= p1(0);
            p1(2) <= p1(1);
            p1(3) <= p1(2);
            p1(4) <= p1(3);
            p1(5) <= p1(4);
            p1(6) <= p1(5);
            p1(7) <= p1(6);

            p2(0) <= so(2)(0);
            p2(1) <= p2(0);
            p2(2) <= p2(1);
            p2(3) <= p2(2);
            p2(4) <= p2(3);
            p2(5) <= p2(4);

            p3(0) <= so(3)(0);
            p3(1) <= p3(0);
            p3(2) <= p3(1);
            p3(3) <= p3(2);

            p4(0) <= so(3)(1);
            p4(1) <= p4(0);
            p4(2) <= p4(1);

            p5(0) <= so(3)(2);
            p5(1) <= p5(0);


         end if;
      end if;


end process;
        ai(0)(0) <= a(0);
        ai(0)(1) <= a1(1);
         ai(0)(2) <= a2(2);
         ai(0)(3) <= a3(3);

         bi(0)(0) <= b(0);
         bi(1)(0) <= b1(1);
         bi(2)(0) <= b2(4);
         bi(3)(0) <= b3(6);
```

```
   p <= co(3)(3) & so(3)(3) & p5(1) & p4(2) & p3(3) & p2(5) & p1(7)
& p0(10);
end structural;
```

Διάγραμμα Δομής



Προσομοίωση

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

entity multiplier_tb is
end entity;

architecture bench of multiplier_tb is

    component multiplier is
        port (  clk,rst: in STD_LOGIC;
            a, b : in STD_LOGIC_VECTOR(3 downto 0);
            p : out STD_LOGIC_VECTOR(7 downto 0)
            );
    end component;

    signal              a :  STD_LOGIC_VECTOR (3 downto 0):=
(others => '0');
    signal              b : STD_LOGIC_VECTOR (3 downto 0) :=
(others => '0');
    signal              p : STD_LOGIC_VECTOR (7 downto 0) ;
    signal              clk : STD_LOGIC;
    signal              rst : STD_LOGIC;
    constant clock_period : time := 10ns;

begin

    uut : multiplier
        port map (
                    a => a,
                    b => b,
                    p => p,
                    clk =>clk,
                    rst => rst
                     );

stimulus : process
```

```
    begin
    rst <= '1';
    a  <= "0000";
    b  <= "0000";
    wait for clock_period*10 ;
    rst <= '0';
    a <= "1111";
    b <= "1111";
    wait;


    end process;

    gen_clock : process
      begin
            clk <= '1';
            wait for clock_period/2;
            clk <= '0';
            wait for clock_period/2;
       end process;

end architecture;
```

Κρίσιμο Μονοπάτι

Το κρίσιμο μονοπάτι του κυκλώματος (ή ένα εξ αυτών με ίση καθυστέρηση) είναι το μονοπάτι p1_reg[7]/C – p[1], με συνολική καθυστέρηση 4.112 ns.

Κατανάλωσης Πόρων

Resource	Utilization	Available	Utilization %
Slice LUTs	45	17600	0.26
Slice Registers	204	35200	0.58
IO	18	102	17.65
Clocking	1	32	3.12

Ερωτήματα Θεωρίας

I. Ποια είναι η διαφορά ενός ακολουθιακού κυκλώματος που έχει σύγχρονα σήματα ελέγχου σε σχέση με το ίδιο ακολουθιακό κύκλωμα που έχει ασύγχρονα σήματα ελέγχου (π.χ reset, enable, load, etc.); Πώς διαφοροποιείται η λειτουργία τους;

Τα ασύγχρονα σήματα ελέγχου επιδρούν στο κύκλωμα οποιαδήποτε στιγμή ενώ τα σύγχρονα μόνο κατά τον χτύπο του ρολογιού (θετική ή αρνητική ακμή του, ανάλογα με τι λογική ακολουθεί το κύκλωμα). Έτσι, ένα κύκλωμα με ασύγχρονα σήματα θα επιτελέσει τις αντίστοιχες λειτουργίες άμεσα ενώ κύκλωμα με σύγχρονα σήματα πρέπει να περιμένει μέχρι το χτύπο του ρολογιού και αν το σήμα αλλάξει μέχρι τότε μπορεί να μην επιδράσει καθόλου.

II. Για ποιο λόγο πραγματοποιείται η σύνθεση του κυκλώματος αφού ο σχεδιαστής έχει υλοποιήσει το κύκλωμα του με VHDL και έχει ελέγξει τη ορθή λειτουργία του με χρήση testbench;Τι παίρνει ως είσοδο το στάδιο της σύνθεσης και τι παράγει ως έξοδο;

Η λειτουργικότητα κάθε module ελέγχεται με την χρήση των testbenches. Για να μπορέσουμε όμως να δούμε όλες τις διασυνδέσεις μεταξύ των διαφόρων modules χρειάζεται να πραγματοποιηθεί η σύνθεση του κυκλώματος. Έπειτα διακρίνονται και τα μονοπάτια π.χ. που οδηγούν την είσοδο *x* στην έξοδο *y* και κατά συνέπεια έχουμε γνώση και του μονοπατιού με την μεγαλύτερη συνολική καθυστέρηση.
Το στάδιο της σύνθεσης παίρνει ως είσοδο τον κώδικα VHDL και παράγει το netlist.

III. Να επισημάνετε και να δικαιολογήσετε τις διαφορές που υπάρχουν μεταξύ ενός testbench για συνδυαστικό κύκλωμα και ενός testbench για ακολουθιακό κύκλωμα.

Η βασική διαφορά έγκειται στην αναγκαιότητα ύπαρξης ρολογιού στα testbench ακολουθιακών κυκλωμάτων και το οποίο πρέπει να προσομοιώσουμε, ενώ δεν υπάρχει στα συνδυαστικά. Επιπροσθέτως, τα ακολουθιακά κυκλώματα απαιτούν reset.

IV. Να σχολιάσετε και να αιτιολογήστε τις διαφορές που παρατηρείτε στο κρίσιμο μονοπάτι και στην κατανάλωση πόρων του FPGA για τα κυκλώματα που υλοποιήθηκαν στο ζητούμενο 3.

Παρατηρούμε ότι το συνδυαστικό κύκλωμα έχει μεγαλύτερο κρίσιμο μονοπάτι και χρησιμοποιεί λιγότερους πόρους από το αντίστοιχο ακολουθιακό. Αυτό συμβαίνει διότι το ακολουθιακό έχει τους επιπλέον καταχωρητές, δηλαδή παραπάνω πόρους, οι οποίοι καθιστούν το κρίσιμο μονοπάτι μεταξύ δύο διαδοχικά flip-flop ή μεταξύ εισόδου/εξόδου και του αμέσως επόμενου/προηγούμενου flip-flop, το οποίο είναι μικρότερο από ένα μονοπάτι από είσοδο σε έξοδο όπως είναι στο συνδυαστικό κύκλωμα.