3η εργαστηριακή άσκηση

Φωτεινή Κυριακοπούλου 03120182
Ιωάννης Μπουφίδης 03120162

Σύγχρονος Full Adder - FA

Υλοποιούμε το full adder σε behavioral αρχιτεκτονική ως εξής:

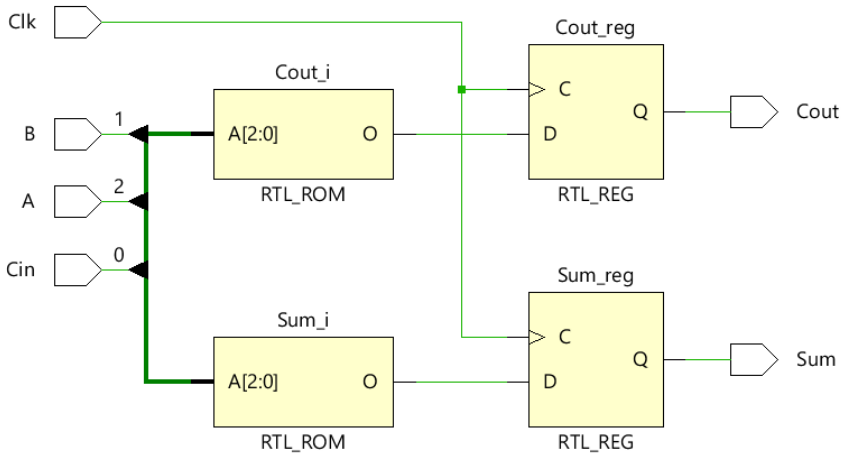
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder is
   Port (
      A, B, Cin, Clk: in std_logic;
      Sum, Cout: out std_logic
   );
end full_adder;

architecture fa_arch of full_adder is
 signal con: std_logic_vector(2 downto 0);
begin
 con <= A&B&Cin;
 process(Clk)
     begin
      if (Clk'event and Clk='1') then
        case con is
      when "000" =>
         Sum <= '0'; Cout <= '0';
      when "001" =>
         Sum <= '1'; Cout <= '0';
      when "010" =>
         Sum <= '1'; Cout <= '0';
      when "011" =>
         Sum <= '0'; Cout <= '1';
      when "100" =>
         Sum <= '1'; Cout <= '0';
      when "101" =>
         Sum <= '0'; Cout <= '1';
      when "110" =>
         Sum <= '0'; Cout <= '1';
      when "111" =>
         Sum <= '1'; Cout <= '1';
      when others =>
         Sum <= 'U'; Cout <= 'U';
     end case;
    end if;
 end process;
end fa_arch;
```

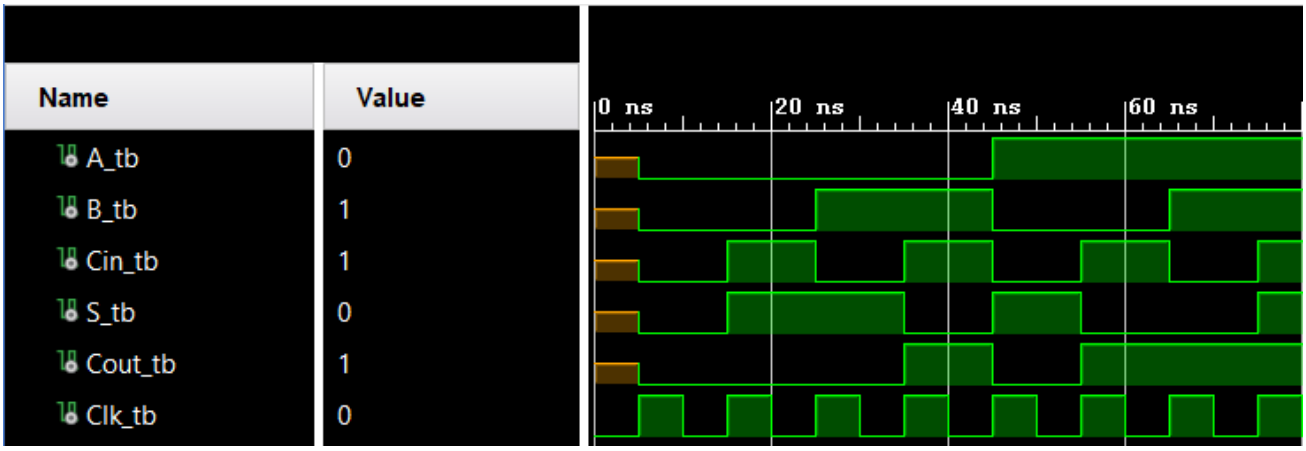
Προκειμένου να προσομοιώσουμε την αρχιτεκτονική, γράφουμε το παρακάτω testbench:

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

entity full_adder_tb is
end full_adder_tb;

architecture full_adder_test of full_adder_tb is
   component full_adder is
      Port (
             A, B, Cin, Clk: in std_logic;
             Sum, Cout: out std_logic
            );
   end component;
signal A_tb, B_tb, Cin_tb, S_tb, Cout_tb, Clk_tb: std_logic;
begin
   uut: full_adder port map (A => A_tb, B => B_tb, Cin => Cin_tb, Sum => S_tb,
Cout => Cout_tb, Clk => Clk_tb);
   check: process
   begin
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '0'; B_tb <= '0'; Cin_tb <= '0';
Clk_tb <= '1'; wait for 5 ns;
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '0'; B_tb <= '0'; Cin_tb <= '1';
Clk_tb <= '1'; wait for 5 ns;
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '0'; B_tb <= '1'; Cin_tb <= '0';
Clk_tb <= '1'; wait for 5 ns;
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '0'; B_tb <= '1'; Cin_tb <= '1';
Clk_tb <= '1'; wait for 5 ns;
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '1'; B_tb <= '0'; Cin_tb <= '0';
Clk_tb <= '1'; wait for 5 ns;
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '1'; B_tb <= '0'; Cin_tb <= '1';
Clk_tb <= '1'; wait for 5 ns;
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '1'; B_tb <= '1'; Cin_tb <= '0';
Clk_tb <= '1'; wait for 5 ns;
      Clk_tb <= '0'; wait for 5 ns; A_tb <= '1'; B_tb <= '1'; Cin_tb <= '1';
Clk_tb <= '1'; wait for 5 ns;
   end process check;
end full_adder_test;
```

3η εργαστηριακή άσκηση

Παρακάτω φαίνεται το αποτέλεσμα της προσομοίωσης (κάτω) και το σχηματικό RTL (πάνω):





Το κρίσιμο μονοπάτι του κυκλώματος είναι **From A To Sum** με χρονική καθυστέρηση **1.932+4.076=6.008 ns**

3η εργαστηριακή άσκηση

Σύγχρονος Αθροιστής διάδοσης κρατουμένου των 4 bits – 4-bit PA

Υλοποιούμε το 4-bit parallel adder με την τεχνική Pipeline (χρησιμοποιώντας FAs) ως εξής:

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

entity parallel_adder is
   port (
      A, B: in std_logic_vector(3 downto 0);
      Cin, Cin: in std_logic;
      Cout: out std_logic;
      Sum: out std_logic_vector(3 downto 0)
   );
end parallel_adder;

architecture pipeline_pa of parallel_adder is
  component full_adder is
    Port (
       A, B, Cin, Clk: in std_logic;
       Sum, Cout: out std_logic
    );
  end component;

    signal A2, B2: std_logic_vector(2 downto 0);
    signal A3, B3: std_logic_vector(1 downto 0);
    signal A4, B4, C2, C3, C4, S1: std_logic;
    signal S2: std_logic_vector(1 downto 0);
    signal S3: std_logic_vector(2 downto 0);
begin
    process(Clk)
    begin
        if (Clk'event and Clk='1') then
            A2 <= A(3 downto 1);
            A3 <= A2(2 downto 1);
            A4 <= A3(1);
            B2 <= B(3 downto 1);
            B3 <= B2(2 downto 1);
            B4 <= B3(1);
            S2(0) <= S1;
            S3(1 downto 0) <= S2;
            Sum(2 downto 0) <= S3;
        end if;
    end process;
```

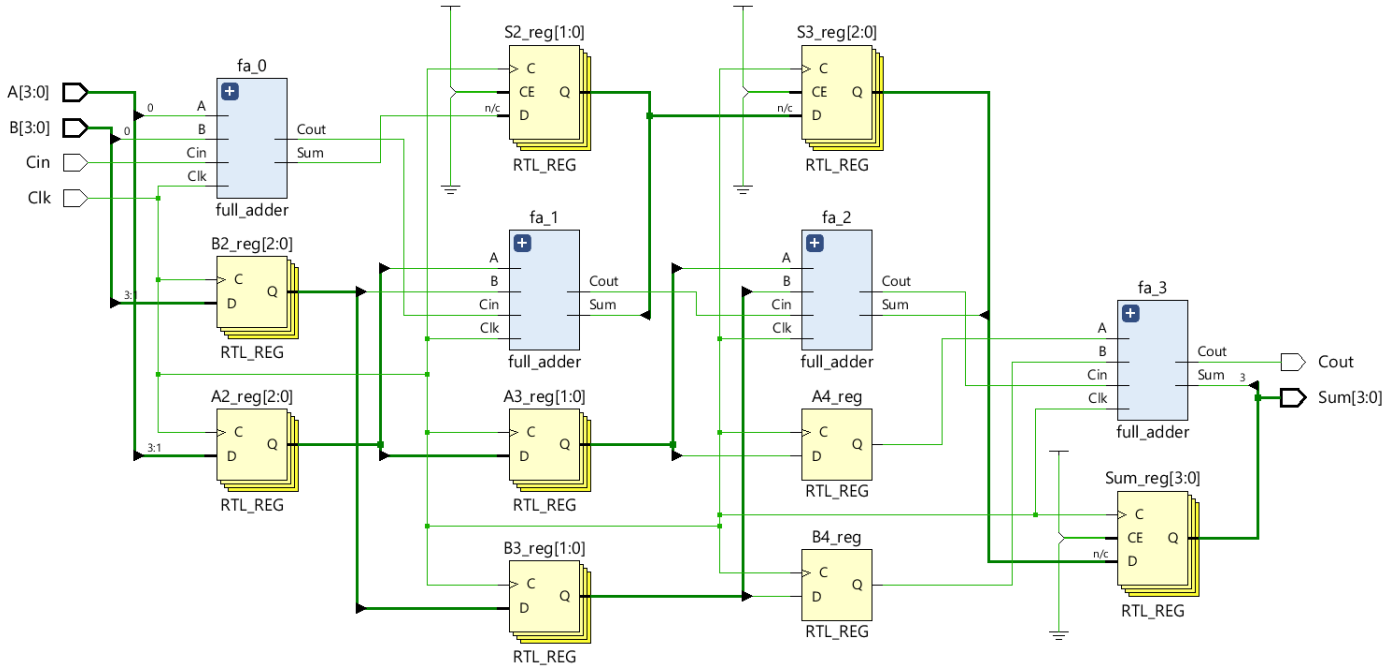
```
    fa_0: full_adder port map(
        A => A(0), B => B(0), Cin => Cin,
        Sum => S1, Cout => C2, Clk => Clk
    );
    fa_1: full_adder port map(
        A => A2(0), B => B2(0), Cin => C2,
        Sum => S2(1), Cout => C3, Clk => Clk
    );
    fa_2: full_adder port map(
        A => A3(0), B => B3(0), Cin => C3,
        Sum => S3(2), Cout => C4, Clk => Clk
    );
    fa_3: full_adder port map(
        A => A4, B => B4, Cin => C4,
        Sum => Sum(3), Cout => Cout, Clk => Clk
    );
end pipeline_pa;
```

Προκειμένου να προσομοιώσουμε την αρχιτεκτονική, γράφουμε το παρακάτω testbench:

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

entity parallel_adder_tb is
end parallel_adder_tb;

architecture parallel_adder_test of parallel_adder_tb is
   component parallel_adder
     Port (
        A, B: in std_logic_vector(3 downto 0);
        Clk, Cin: in std_logic;
        Sum: out std_logic_vector(3 downto 0);
        Cout: out std_logic
      );
   end component;
   signal A_tb, B_tb, Sum_tb: std_logic_vector(3 downto 0);
   signal Cin_tb, Cout_tb, Clk_tb: std_logic;
begin
  uut: parallel_adder port map (
    A => A_tb,
    B => B_tb,
    Sum => Sum_tb,
    Clk => Clk_tb,
    Cin => Cin_tb,
    Cout => Cout_tb
  );
```

```
  check: process
   begin
     Cin_tb <= '0';

     Clk_tb <= '0'; A_tb <= "1000"; B_tb <= "0101"; wait for 10 ns;
     Clk_tb <= '1'; wait for 10ns;
     Clk_tb <= '0'; A_tb <= "0010"; B_tb <= "1101"; wait for 10 ns;
     Clk_tb <= '1'; wait for 10ns;
     Clk_tb <= '0'; A_tb <= "1110"; B_tb <= "1111"; wait for 10 ns;
     Clk_tb <= '1'; wait for 10ns;
     Clk_tb <= '0'; A_tb <= "1001"; B_tb <= "1001"; wait for 10 ns;
     Clk_tb <= '1'; wait for 10ns;
     Clk_tb <= '0'; Cin_tb <= '1'; A_tb <= "0110"; B_tb <= "0001"; wait for 10 ns;
     Clk_tb <= '1'; wait for 10ns;
   end process check;
end parallel_adder_test;
```

Το σχηματικό RTL που προκύπτει είναι το εξής:

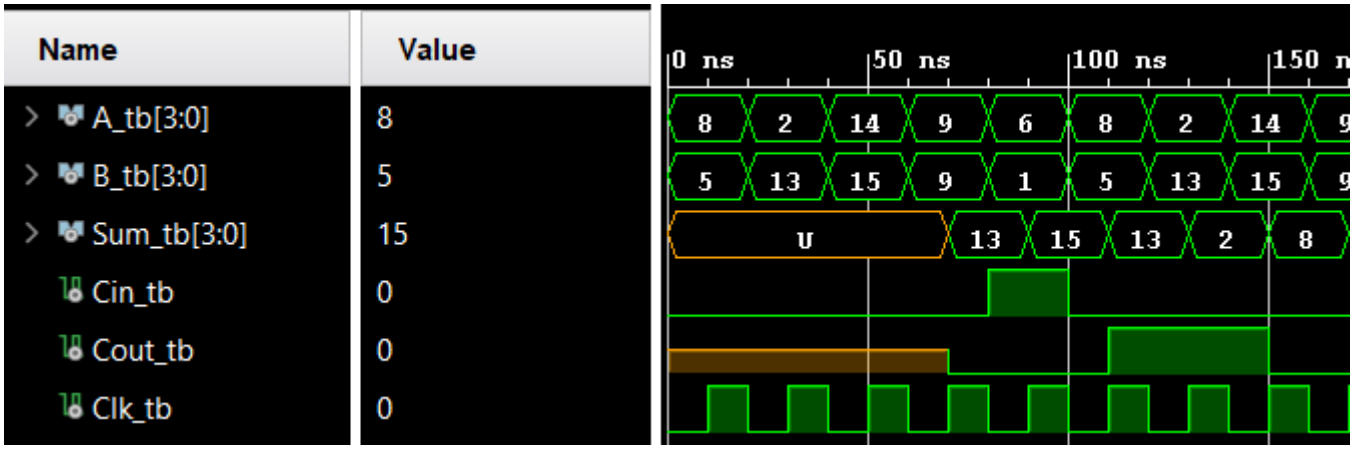


Από το παραπάνω σχήμα παρατηρούμε ότι προσθέσαμε επιπλέον καταχωρητές για την αποθήκευση των εισόδων A και B από το 1ο ως το προτελευταίο (εδώ 3ο) στάδιο της σωλήνωσης, αλλά και για την αποθήκευση της εξόδου Sum από το 2ο ως το τελευταίο στάδιο του pipeline.

Οι καταχωρητές αυτοί τοποθετήθηκαν, με σκοπό το κύκλωμα να έχει την δυνατότητα να τροφοδοτείται σε κάθε κύκλο με καινούρια είσοδο. Συγκεκριμένα, πρέπει σε κάθε επόμενο στάδιο να αποθηκεύονται (και να προωθούνται) τα απαιτούμενα bits των εισόδων και της εξόδου, ώστε το αποτέλεσμα της εξόδου να είναι σωστό χωρίς να επηρεάζεται από τις εισόδους που δέχτηκε το κύκλωμα στους κύκλους (3 κύκλοι) που παρεμβλήθηκαν μέχρι τον υπολογισμό του.

3η εργαστηριακή άσκηση

Παρακάτω φαίνεται το αποτέλεσμα της προσομοίωσης:



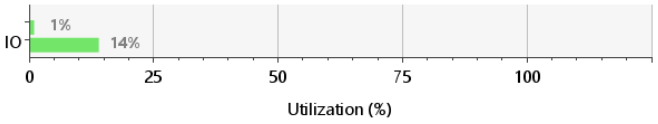
Το κρίσιμο μονοπάτι του κυκλώματος είναι **From Input To Cout** με χρονική καθυστέρηση **4.076**.

Ο αντίστοιχος ασύγχρονος αθροιστής 4 bits της 2ης Εργαστηριακής Άσκησης είχε χρονική καθυστέρηση **5.377**, γεγονός που σημαίνει ότι το pipeline μειώνει το critical path, άρα και την καθυστέρηση του κυκλώματος. Με την τεχνική της σωλήνωσης υπάρχει η αρχική καθυστέρηση των 3 κύκλων μέχρι να παραχθεί το πρώτο άθροισμα. Ωστόσο, για μεγαλό πλήθος πράξεων αυξάνεται η απόδοση.

Κατανάλωση πόρων FPGA	Κατανάλωση Ισχύος

Ασύγχρονος 4-bit Παράλληλος Αθροιστής

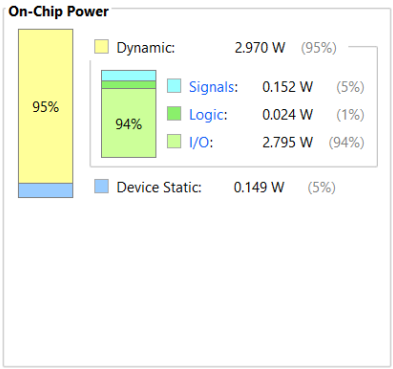
Resource	Utilization	Available	Utilization %
LUT	4	17600	0.02
IO	14	100	14.00



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

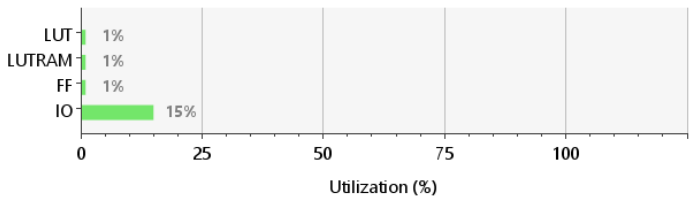
Total On-Chip Power:	**3.12 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**61,0°C**
Thermal Margin:	24,0°C (2,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Σύγχρονος 4-bit Παράλληλος Αθροιστής με τεχνική Pipeline

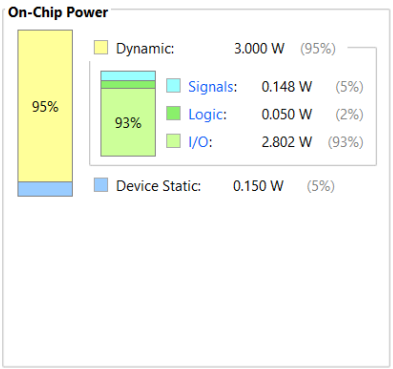
Resource	Utilization	Available	Utilization %
LUT	6	17600	0.03
LUTRAM	2	6000	0.03
FF	21	35200	0.06
IO	15	100	15.00



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**3.15 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**61,3°C**
Thermal Margin:	23,7°C (2,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

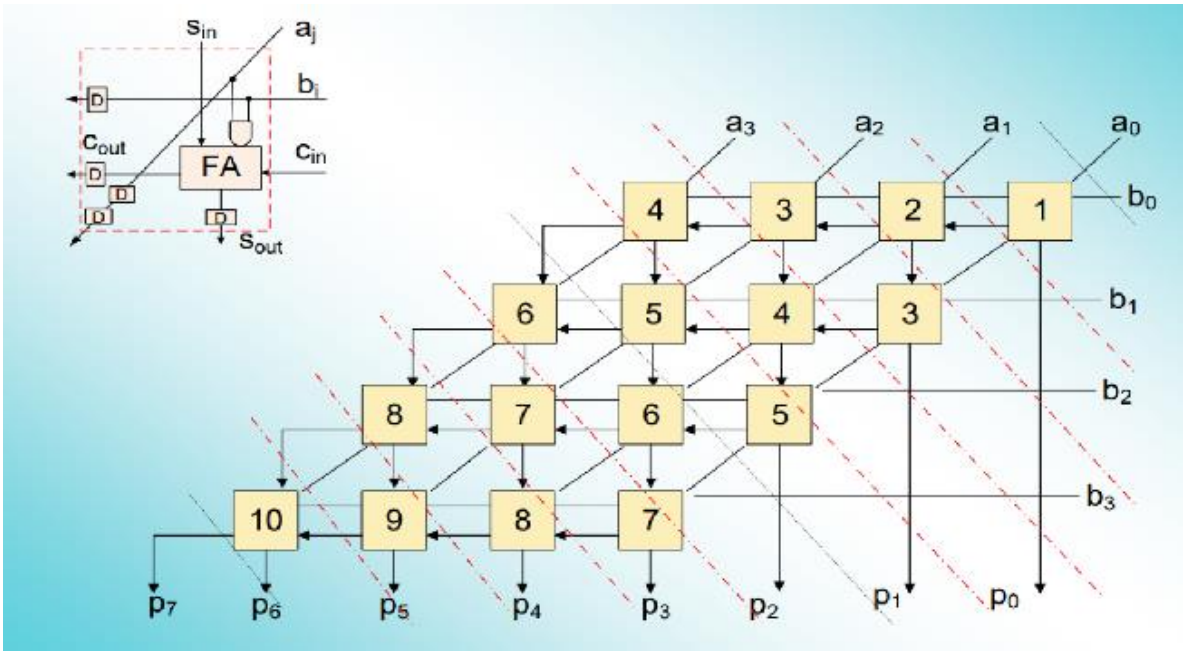


Από τον παραπάνω πίνακα παρατηρούμε ότι η αρχιτεκτονική με το pipeline οδηγεί σε μεγαλύτερη κατανάλωση απ’ ότι αυτή χωρίς το pipeline. Αυτό συμβαίνει, διότι στην πρώτη χρησιμοποιούμε παραπάνω D Flip-Flops προκειμένου να συγχρονίσουμε το pipeline.

3η εργαστηριακή άσκηση

Συστολικός 4-bit Πολλαπλασιαστής διάδοσης κρατουμένων

Για την περιγραφή του πολλαπλασιαστή βασιστήκαμε στο παρακάτω σχήμα:



Προκειμένου να περιγράψουμε το παραπάνω κύκλωμα με Structural αρχιτεκτονική, δημιουργήσαμε τις εξής οντότητες:

- **D Flip-Flops** (καθυστέρησης 1,2,3 ή 4 κύκλων)

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

entity d1 is
    Port(
          D, Clk: in std_logic;
          Q: out std_logic
         );
end d1;

architecture behavioral of d1 is
begin
    process(Clk)
    begin
      if (Clk'event and Clk='1') then
            Q <= D;
      end if;
    end process;
end behavioral;
```

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

entity d2 is
    Port(
          D, Clk: in std_logic;
          Q: out std_logic
         );
end d2;

architecture structural of d2 is
    component d1 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
             );
    end component;
    signal tmp: std_logic;
begin
    l1: d1 port map (D => D, Q => tmp, Clk => Clk);
    l2: d1 port map (D => tmp, Q => Q, Clk => Clk);
end structural;
```

Αντίστοιχα, υλοποιούμε τα Flip-Flops με καθυστέρηση 3 και 4 κύκλων.

- **Κύταρρο του πολλαπλασιαστή** (όπως φαίνεται πάνω αριστερά στην παραπάνω εικόνα)

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

entity mul_cell is
    Port (
        A1, A2, B, Cin, Clk: in std_logic;
        P, Cout, A_out, B_out: out std_logic
     );
end mul_cell;

architecture mul_cell_arch of mul_cell is
    component d1 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    component d2 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    component full_adder is
      Port (
           A, B, Cin, Clk: in std_logic;
           Sum, Cout: out std_logic
           );
    end component;
    signal and1: std_logic;
begin
    and1 <= A1 and A2;
    c1: full_adder port map (A => and1, B => B, Cin => Cin, Clk => Clk,
                             Sum => P, Cout => Cout);
    de1: d1 port map (D => A2, Q => B_out, Clk => Clk);
    de2: d2 port map (D => A1, Q => A_out, Clk => Clk);
end mul_cell_arch;
```

- **Μονάδα καθυστέρησης εισόδου για το πρώτο στάδιο του pipeline**

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


entity init_delay is
  Port (
        A: in std_logic_vector(3 downto 0);
        Clk: in std_logic;
        B: out std_logic_vector(3 downto 0)
  );
end init_delay;

architecture structural of init_delay is
    component d1 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    component d2 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    component d3 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    component d4 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;
begin
    f1: d1 port map (D => A(0), Q => B(0), Clk => Clk);
    f2: d2 port map (D => A(1), Q => B(1), Clk => Clk);
    f3: d3 port map (D => A(2), Q => B(2), Clk => Clk);
    f4: d4 port map (D => A(3), Q => B(3), Clk => Clk);
end structural;
```

3η εργαστηριακή άσκηση

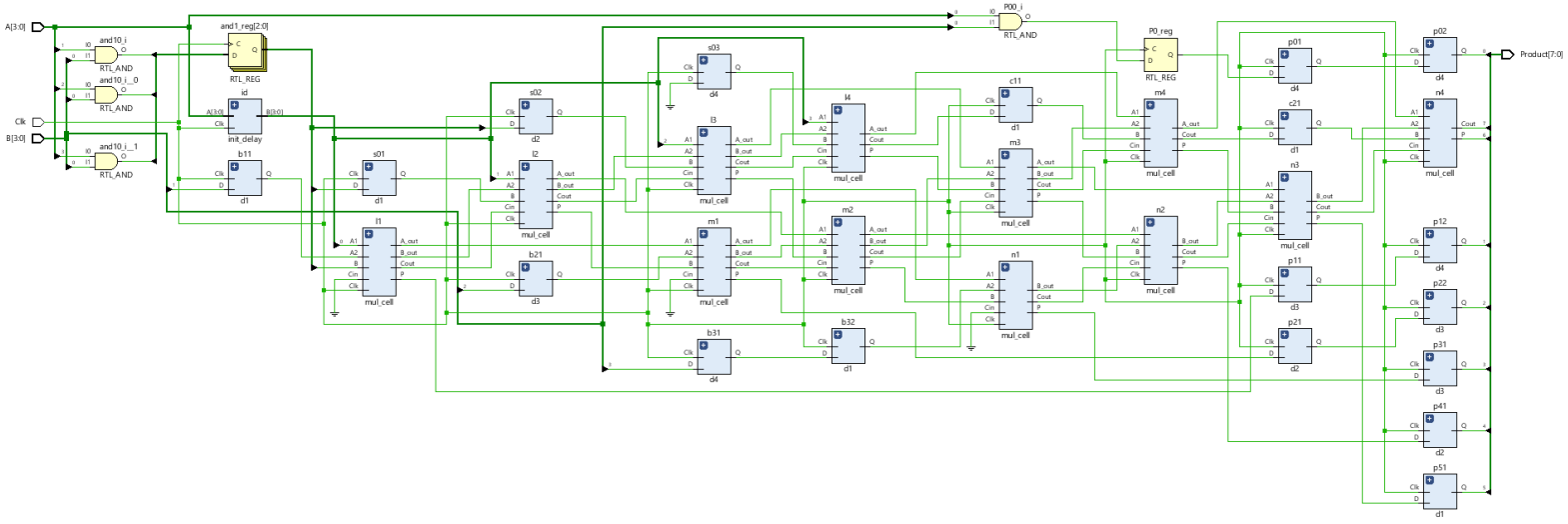
Χρησιμοποιώντας τις παραπάνω οντότητες και προσθέτοντας τις κατάλληλες καθυστέρησεις όπου χρειαζόταν (πχ διαγώνια carries, bits του B, bits του γινομένου κτλ), προκύπτει η εξής περιγραφή του 4-bit συστολικού πολλαπλασιαστή:

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

entity multiplier is
    Port (
        A, B : in STD_LOGIC_VECTOR(3 downto 0);
        Clk: in STD_LOGIC;
        Product : out STD_LOGIC_VECTOR(7 downto 0)
    );
end multiplier;

architecture multiplier_arch of multiplier is
    component mul_cell is
        Port (
            A1, A2, B, Cin, Clk: in std_logic;
            P, Cout, A_out, B_out: out std_logic
             );
    end component;

    component init_delay is
        Port (
            A: in std_logic_vector(3 downto 0);
            Clk: in std_logic;
            B: out std_logic_vector(3 downto 0)
             );
    end component;

    component d1 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    component d2 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    component d3 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;
```

```
    component d4 is
        Port(
              D, Clk: in std_logic;
              Q: out std_logic
            );
    end component;

    signal A1, A2, A3, B1, B2, B3: std_logic_vector(3 downto 0);
    signal S1, S2, C1, C2, C3: std_logic_vector(3 downto 0);
    signal and1, S0: std_logic_vector(2 downto 0);
    signal P0, P0_d, P1, P1_d, P2, P2_d, P3, P4, P5: std_logic;
    signal B3_d: std_logic;
begin
    process(Clk)
    begin
        if (Clk'event and Clk='1') then
            and1 <= (A(3) and B(0))&(A(2) and B(0))&(A(1) and B(0));
            P0 <= A(0) and B(0);
        end if;
    end process;
    --synch entity for 1st stage's inputs
    id: init_delay port map (A => A, B => A1, Clk => Clk);

    b11: d1 port map (D => B(1), Q => B1(0), Clk => Clk);
    b21: d3 port map (D => B(2), Q => B2(0), Clk => Clk);
    b31: d4 port map (D => B(3), Q => B3_d, Clk => Clk); b32: d1 port map (D =>
B3_d, Q => B3(0), Clk => Clk);

    --final products' delay
    p01: d4 port map (D => P0, Q => P0_d, Clk => Clk); p02: d4 port map (D => P0_d,
Q => Product(0), Clk => Clk);
    p11: d3 port map (D => P1, Q => P1_d, Clk => Clk); p12: d4 port map (D => P1_d,
Q => Product(1), Clk => Clk);
    p21: d2 port map (D => P2, Q => P2_d, Clk => Clk); p22: d3 port map (D => P2_d,
Q => Product(2), Clk => Clk);

    s01: d1 port map (D => and1(1), Q => S0(0), Clk => Clk);
    s02: d2 port map (D => and1(2), Q => S0(1), Clk => Clk);
    s03: d4 port map (D => '0', Q => S0(2), Clk => Clk);

    --1st stage
    l1: mul_cell port map(A1 => A1(0), A2 => B1(0), B => and1(0), Cin => '0', Clk
=> Clk, P => P1, Cout => C1(0), B_out => B1(1), A_out => A2(0));
    l2: mul_cell port map(A1 => A1(1), A2 => B1(1), B => S0(0), Cin => C1(0), Clk
=> Clk, P => S1(0), Cout => C1(1), B_out => B1(2), A_out => A2(1));
    l3: mul_cell port map(A1 => A1(2), A2 => B1(2), B => S0(1), Cin => C1(1), Clk
=> Clk, P => S1(1), Cout => C1(2), B_out => B1(3), A_out => A2(2));
    l4: mul_cell port map(A1 => A1(3), A2 => B1(3), B => S0(2), Cin => C1(2), Clk
=> Clk, P => S1(2), Cout => C1(3), A_out => A2(3));
```

```
    --2nd stage
    c11: d1 port map (D => C1(3), Q => S1(3), Clk => Clk);
    m1: mul_cell port map(A1 => A2(0), A2 => B2(0), B => S1(0), Cin => '0', Clk =>
Clk, P => P2, Cout => C2(0), B_out => B2(1), A_out => A3(0));
    m2: mul_cell port map(A1 => A2(1), A2 => B2(1), B => S1(1), Cin => C2(0), Clk
=> Clk, P => S2(0), Cout => C2(1), B_out => B2(2), A_out => A3(1));
    m3: mul_cell port map(A1 => A2(2), A2 => B2(2), B => S1(2), Cin => C2(1), Clk
=> Clk, P => S2(1), Cout => C2(2), B_out => B2(3), A_out => A3(2));
    m4: mul_cell port map(A1 => A2(3), A2 => B2(3), B => S1(3), Cin => C2(2), Clk
=> Clk, P => S2(2), Cout => C2(3), A_out => A3(3));

    --3rd stage
    c21: d1 port map (D => C2(3), Q => S2(3), Clk => Clk);
    n1: mul_cell port map(A1 => A3(0), A2 => B3(0), B => S2(0), Cin => '0', Clk =>
Clk, P => P3, Cout => C3(0), B_out => B3(1));
    n2: mul_cell port map(A1 => A3(1), A2 => B3(1), B => S2(1), Cin => C3(0), Clk
=> Clk, P => P4, Cout => C3(1), B_out => B3(2));
    n3: mul_cell port map(A1 => A3(2), A2 => B3(2), B => S2(2), Cin => C3(1), Clk
=> Clk, P => P5, Cout => C3(2), B_out => B3(3));
    n4: mul_cell port map(A1 => A3(3), A2 => B3(3), B => S2(3), Cin => C3(2), Clk
=> Clk, P => Product(6), Cout => Product(7));

    --final products
    p31: d3 port map (D => P3, Q => Product(3), Clk => Clk);
    p41: d2 port map (D => P4, Q => Product(4), Clk => Clk);
    p51: d1 port map (D => P5, Q => Product(5), Clk => Clk);


end multiplier_arch;
```

Παρακάτω φαίνεται το σχηματικό RTL του κυκλώματος:



Προκειμένου να ελέγξουμε την σωστή λειτουργία του κυκλώματος, γράφουμε το εξής testbench:

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

entity multiplier_tb is
end multiplier_tb;

architecture multiplier_test of multiplier_tb is
   component multiplier
     Port (
        A, B: in STD_LOGIC_VECTOR(3 downto 0);
        Clk: in STD_LOGIC;
        Product : out STD_LOGIC_VECTOR(7 downto 0)
    );
   end component;
   signal A_tb, B_tb: std_logic_vector(3 downto 0);
   signal P_tb: std_logic_vector(7 downto 0);
   signal Clk_tb: std_logic;
begin
  uut: multiplier port map (
    A => A_tb,
    B => B_tb,
    Product => P_tb,
    Clk => Clk_tb
  );

  check: process
  begin
    Clk_tb <= '0';
    A_tb <= "1000"; B_tb <= "0100"; wait for 10 ns;
    Clk_tb <= '1'; wait for 10 ns;

    Clk_tb <= '0';
    A_tb <= "0011"; B_tb <= "1011"; wait for 10 ns;
    Clk_tb <= '1'; wait for 10 ns;

    Clk_tb <= '0';
    A_tb <= "1100"; B_tb <= "1101"; wait for 10 ns;
    Clk_tb <= '1'; wait for 10ns;

    Clk_tb <= '0';
    A_tb <= "0000"; B_tb <= "0001"; wait for 10 ns;
    Clk_tb <= '1'; wait for 10ns;

    Clk_tb <= '0';
    A_tb <= "1111"; B_tb <= "1111"; wait for 10 ns;
    Clk_tb <= '1'; wait for 10ns;
```

```
    Clk_tb <= '0';
    A_tb <= "1001"; B_tb <= "1111"; wait for 10 ns;
    Clk_tb <= '1'; wait for 10ns;
  end process check;
end multiplier_test;
```

Το αποτέλεσμα της προσομοίωσης φαίνεται στο παρακάτω σχήμα, ενώ το κρίσιμο μονοπάτι του κυκλώματος είναι **από την είσοδο του τελευταίου κυτάρρου του πολλαπλασιαστή** (ή **την είσοδο του τελευταίου FF** αν πρόκειται για το 6ο bit του γινομένου) **μέχρι το 7ο, 8ο ή 6ο bit της εξόδου αντίστοιχα** με χρονική καθυστέρηση **4.076**.