Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ψηφιακά Συστήματα VLSI

3η Εργαστηριακή Άσκηση

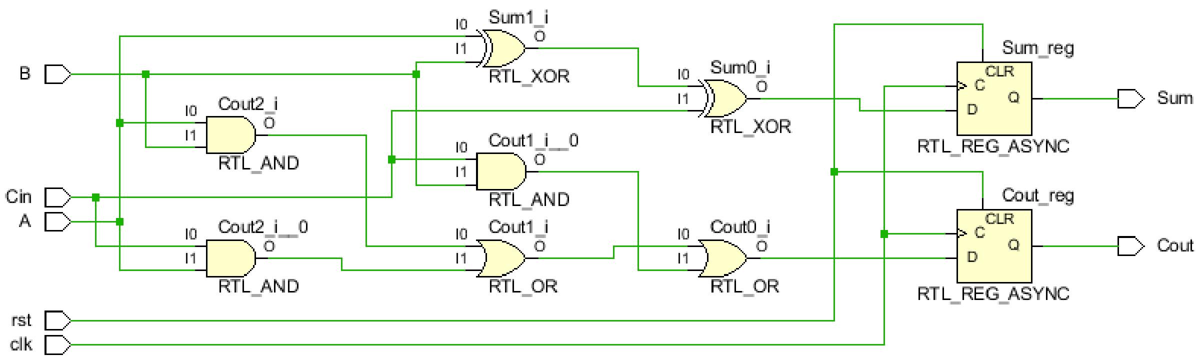
Σχεδίαση Μονάδων Υλικού με την Τεχνική Pipelining

Καραμπίνας Παναγιώτης (03116170)
Αλικάρης Αντώνιος Δημήτριος (03118062)

1. Σύγχρονος Πλήρης Αθροιστής

Κώδικας και δομικό διάγραμμα (RTL)

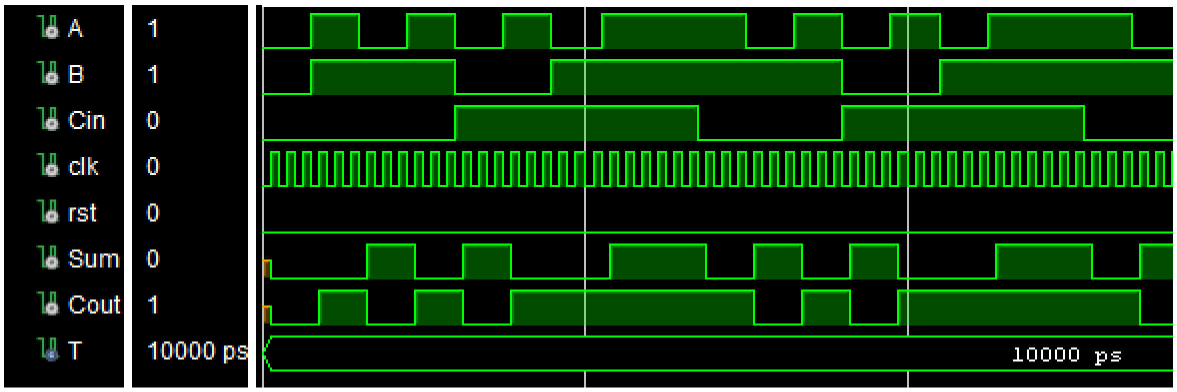
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Sychronous_Full_Adder_Behavioral is
  port(
       clk : in std_logic;
       rst : in std_logic;
       A : in  std_logic;
       B   : in  std_logic;
       Cin : in std_logic;
       Sum   : out  std_logic;
       Cout   : out  std_logic
      );
end entity; -- Synchronous Full Adder

architecture behavioral_arch of Sychronous_Full_Adder_Behavioral is
--In the below implementation of the Sychronous adder we observe that only when
↪  we have a logic switch
--of our clock from 0 -> 1, there happens a change in our FA variables.
--Another point of interest is that the use of variables other than Sum and Cout
↪  inside the if(of the CLK event)
--triggers an unwanted behavior. So we understand that for every variable used
↪  inside the if (CLK event) our compiler
--creates a register, therefore it is logical to use a minimal number of
↪  variables inside the if statement.
begin
  FA_LOGIC : process(rst,clk)
  begin
    if(rst='1') then
        Sum <='0';
        Cout<='0';
    elsif (clk'event and clk='1') then
        Sum <= A XOR B XOR Cin ;
        Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;
    end if;
  end process;

end architecture ; -- arch
```



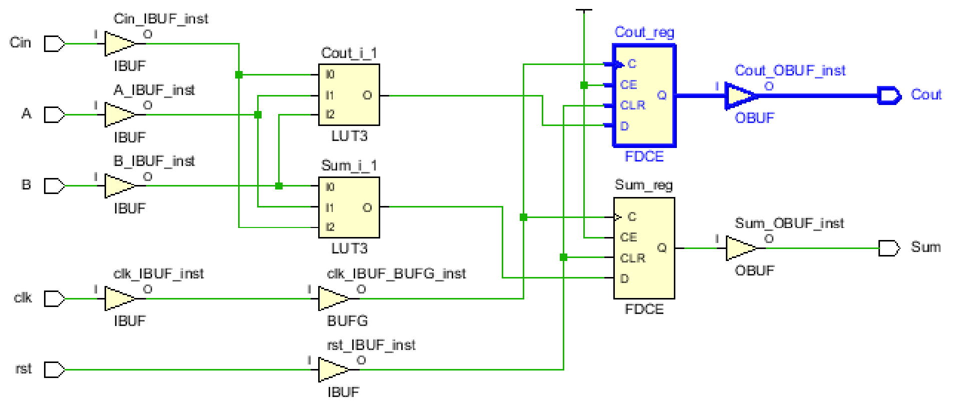
Κώδικας testbench και Κυματομορφή εξόδου

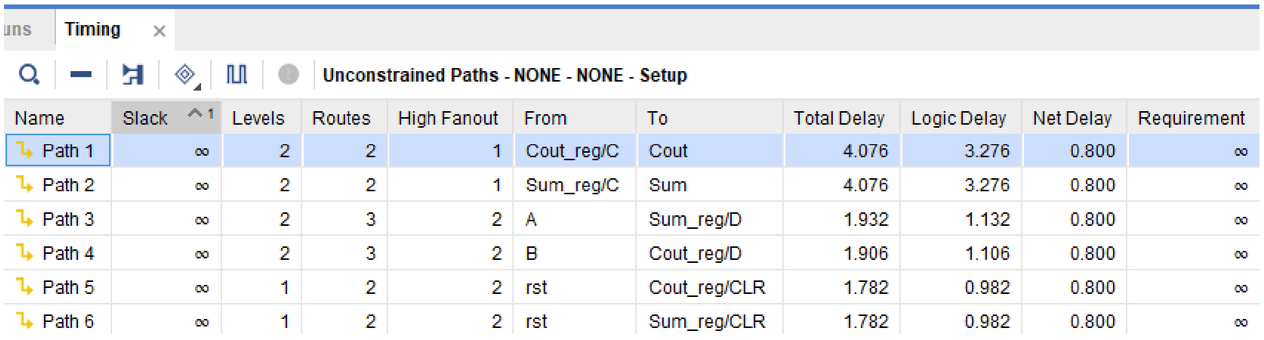
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Sychronous_Full_Adder_Behavioral_tb IS
END Sychronous_Full_Adder_Behavioral_tb;

ARCHITECTURE behavior OF Sychronous_Full_Adder_Behavioral_tb IS

 -- Component Declaration for the Unit Under Test (UUT)

 COMPONENT Sychronous_Full_Adder_Behavioral
 PORT(
 A : IN std_logic;
 B : IN std_logic;
 Cin : IN std_logic;
 clk : in std_logic;
 rst : in std_logic;
 Sum : OUT std_logic;
 Cout : OUT std_logic
 );
 END COMPONENT;

 --constants
 constant T : time := 10ns; --clock period
 -- Inputs
 signal A : std_logic := '0';
 signal B : std_logic := '0';
 signal Cin : std_logic := '0';
 signal clk : std_logic := '0';
 signal rst : std_logic := '0';

 -- Outputs
 signal Sum : std_logic;
 signal Cout : std_logic;

BEGIN

 -- Instantiate the Unit Under Test (UUT)
 uut: Sychronous_Full_Adder_Behavioral PORT MAP (
 A => A,
 B => B,
 Cin => Cin,
 clk => clk,
 rst => rst,
 Sum => Sum,
 Cout => Cout
 );

 clk_gen: process begin
    clk <= '0';
    wait for T/2;
    clk <= '1';
    wait for T/2;
 end process;

 -- Stimulus process
 stim_proc: process
 begin
 -- hold reset state for 100 ns.
 wait for 30 ns;

 -- insert stimulus here
 A <= '1';
 B <= '1';
 Cin <= '0';
 wait for 30 ns;

 A <= '0';
 B <= '1';
 Cin <= '0';
 wait for 30 ns;

 A <= '1';
 B <= '1';
 Cin <= '0';
 wait for 30 ns;

 A <= '0';
 B <= '0';
 Cin <= '1';
 wait for 30 ns;
```

```

A <= '1';
B <= '0';
Cin <= '1';
wait for 30 ns;

A <= '0';
B <= '1';
Cin <= '1';
wait for 30 ns;

A <= '1';
B <= '1';
Cin <= '1';
wait for 30 ns;

end process;

END;
```



Critical Path και Συνολική Καθυστέρηση





Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	2	2	1	Cout_reg/C	Cout	4.076	3.276	0.800	∞
Path 2	∞	2	2	1	Sum_reg/C	Sum	4.076	3.276	0.800	∞
Path 3	∞	2	3	2	A	Sum_reg/D	1.932	1.132	0.800	∞
Path 4	∞	2	3	2	B	Cout_reg/D	1.906	1.106	0.800	∞
Path 5	∞	1	2	2	rst	Cout_reg/CLR	1.782	0.982	0.800	∞
Path 6	∞	1	2	2	rst	Sum_reg/CLR	1.782	0.982	0.800	∞

Όπως φαίνεται από την παραπάνω εικόνα, το critical path είναι από τον τελευταίο καταχωρητή στο Cout και η συνολική καθυστέρηση είναι 4.076ns.

2. Σύγχρονος Αθροιστής Διάδοσης κρατουμένου 4ων bit

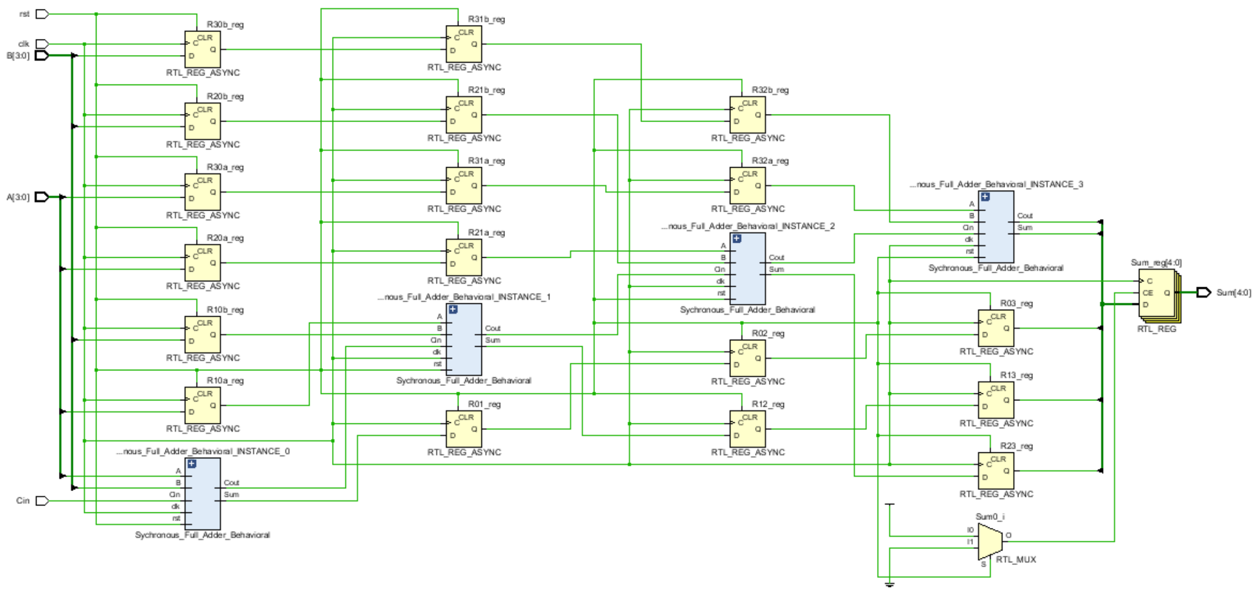
Κώδικας και δομικό διάγραμμα (RTL)

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

entity Sychronous_4bit_Adder_Structural is
  port(
        clk : in  std_logic;
        rst : in  std_logic;
        A   : in  std_logic_vector(3 downto 0);
        B   : in  std_logic_vector(3 downto 0);
        Cin   : in  std_logic;
        Sum    : out std_logic_vector(4 downto 0);
        Cout     : out  std_logic
       );
end entity; -- Sychronous_4bit_Adder_Structural

architecture structural_arch of Sychronous_4bit_Adder_Structural is

  -------------------------------------------
  -- Declarations of lower level components
  -- used in this level of hierarchy

  --Our Implementation is only based on building blocks of the
  ↪ "Sychronous_Full_Adder_Behavioral", we also use
  --flip - flops but not in a structural way. The VLSI design was based on the
  ↪ diagram given below.
  component Sychronous_Full_Adder_Behavioral is
    port(
        clk : in std_logic;
        rst : in std_logic;
        A : in  std_logic;
        B   : in  std_logic;
        Cin : in std_logic;
        Sum   : out  std_logic;
        Cout   : out  std_logic
         );
  end component;

  -------------------------------------
  -- Declarations of internal signals
  -- used in this level of hierarchy
signal C, S : std_logic_vector(3 downto 0);
signal R01, R02, R03, R10a, R10b, R12, R13, R20a, R20b, R21a, R21b, R23, R30a,
↪ R30b, R31a, R31b, R32a, R32b: std_logic := '0';
```

```

--4 stages of sychronous full adders according to our diagram
begin
  -------------------------------------
  -- LEVEL 0 component instantiation --
  -------------------------------------
  Sychronous_Full_Adder_Behavioral_INSTANCE_0 : Sychronous_Full_Adder_Behavioral
    port map (
            clk => clk,
            rst => rst,
            A   => A(0),
            B   => B(0),
            Cin => Cin,
            Sum =>  S(0),
            Cout =>  C(0)
       );
  -------------------------------------
  -- LEVEL 1 component instantiation --
  -------------------------------------


  Sychronous_Full_Adder_Behavioral_INSTANCE_1 : Sychronous_Full_Adder_Behavioral
    port map (
            clk => clk,
            rst => rst,
            A   => R10a,
            B   => R10b,
            Cin => C(0),
            Sum => S(1),
            Cout => C(1)
             );
  -------------------------------------
  -- LEVEL 2 component instantiation --
  -------------------------------------
  Sychronous_Full_Adder_Behavioral_INSTANCE_2 : Sychronous_Full_Adder_Behavioral
    port map (
            clk => clk,
            rst => rst,
            A   => R21a,
            B   => R21b,
            Cin => C(1),
            Sum => S(2),
            Cout => C(2)
             );
  -------------------------------------
  -- LEVEL 3 component instantiation --
  -------------------------------------
  Sychronous_Full_Adder_Behavioral_INSTANCE_3 : Sychronous_Full_Adder_Behavioral
```

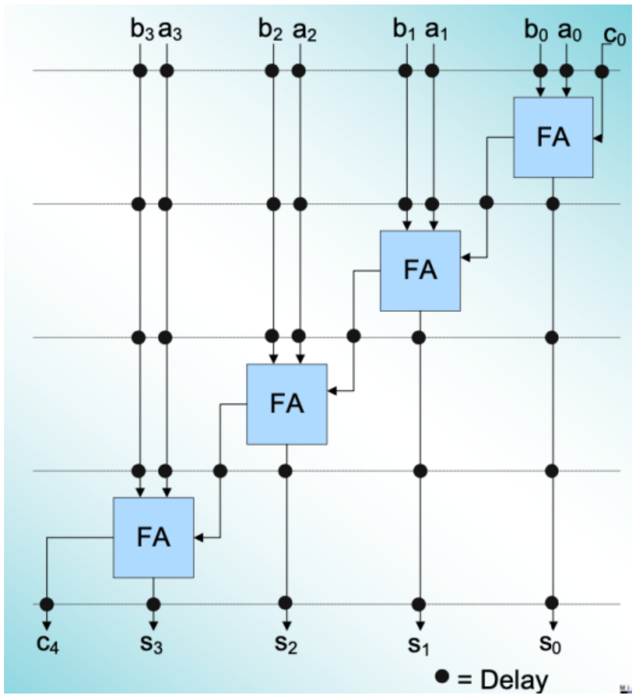
```
    port map (
            clk => clk,
            rst => rst,
            A   => R32a,
            B   => R32b,
            Cin => C(2),
            Sum => S(3),
            Cout => C(3)
             );

  --pipelining procedure:
  --  We made a single if statement for every single flip - flop we want to
  ↪  create. Not the best procedure but it works fine.
  --  By creating a flip - flop structure the code would be more organized
  ↪  and easy to read. Comments in the below statement
  --  explain the use of our flip - flops in respect to the given diagram.
  pipeline_proc: process(clk, rst)
  begin
    if (rst = '1') then

        R01 <= '0'; R02 <= '0'; R03 <= '0';
        R10a <= '0'; R10b <= '0'; R12 <= '0'; R13 <= '0';
        R20a <= '0'; R20b <= '0'; R21a <= '0'; R21b <= '0';
        R23 <= '0'; R30a <= '0'; R30b <= '0';
        R31a <= '0'; R31b <= '0'; R32a <= '0'; R32b <= '0';

    elsif (clk'event and clk='1') then
        --delays for the s0 column
        --After FA
        Sum(0) <= R03; R03 <= R02; R02 <= R01; R01 <= S(0);



        --delays for the Sum1 column
        --After FA
        Sum(1) <= R13; R13 <= R12; R12 <= S(1);
        --Before FA
        R10a <= A(1);
        R10b <= B(1);



        --delays for the Sum2 column
        --After FA
        Sum(2) <= R23; R23 <= S(2);
        --Before FA
        R21a <= R20a; R20a <= A(2);
        R21b <= R20b; R20b <= B(2);
```

```


            --delays for the Sum3 column
            --After FA
            Sum(3) <= S(3);
            --Before FA
            R32a <= R31a; R31a <= R30a; R30a <= A(3);
            R32b <= R31b; R31b <= R30b; R30b <= B(3);

            --delays for the Sum4 column
            Sum(4) <= C(3);
        end if;
      end process;
end architecture ; -- arch
```



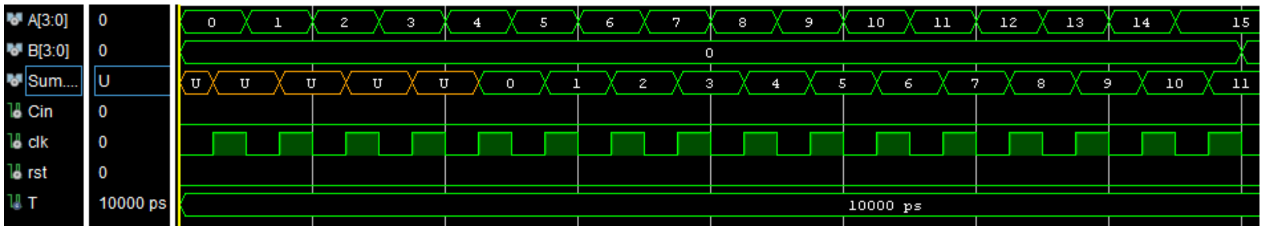
Προσθέσαμε επιπλέον καταχωρητές πριν και μετά τα full adder blocks, τόσους ώστε τα δεδομένα να είναι στο ίδιο επίπεδο την ίδια χρονική στιγμή. Για παράδειγμα, όταν ο πρώτος αθροιστής υπολογίσει το αποτέλεσμα, αυτό το εμφανιστεί στην έξοδο μετά από 3 καθυστερήσεις και αυτό διότι ο τελευταίος αθροιστής θα υπολογίσει τον ίδιο αριθμό 3 κύκλους μετά τον 1ο.

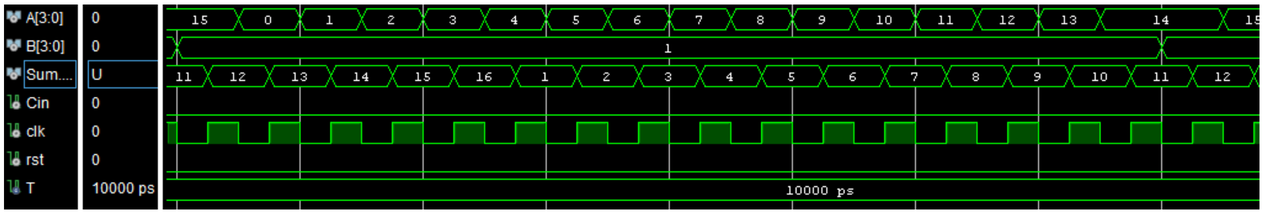
Πιο συγκεκριμένα, οι καθυστερήσεις σε κάθε στάδιο έγιναν με βάση το παρακάτω διάγραμμα, όπου στα σημεία που υπάρχουν κουκκίδες προστέθηκε μία καθυστέρηση δηλαδή ένας καταχωρητής.

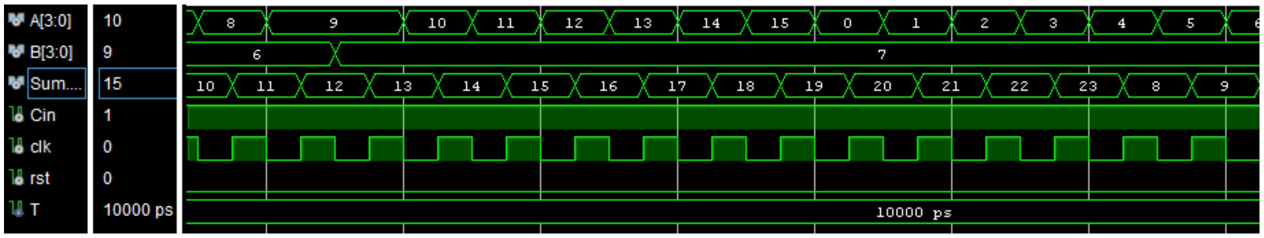


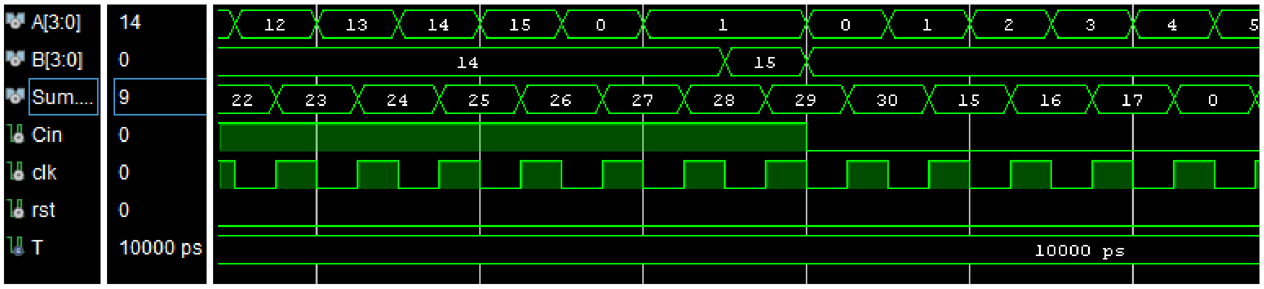
Κώδικας testbench και Κυματομορφή εξόδου

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.std_logic_unsigned.ALL;


entity Sychronous_4bit_Adder_Structural_tb is
--  Port ( );
end entity;

architecture Bench of Sychronous_4bit_Adder_Structural_tb is

COMPONENT Sychronous_4bit_Adder_Structural is
    Port (
        clk : in  std_logic;
        rst : in  std_logic;
        A   : in  std_logic_vector(3 downto 0);
        B   : in  std_logic_vector(3 downto 0);
        Cin   : in  std_logic;
        Sum    : out std_logic_vector(4 downto 0)
            );
end COMPONENT;

SIGNAL A, B : STD_LOGIC_VECTOR(3 downto 0);
SIGNAL Sum : STD_LOGIC_VECTOR(4 downto 0);
SIGNAL Cin : STD_LOGIC := '0';
SIGNAL clk :  std_logic;
SIGNAL rst :  std_logic := '0';

CONSTANT T : TIME := 10 ns;

begin

uut: Sychronous_4bit_Adder_Structural PORT MAP (
                                    clk =>clk,
                                    rst => rst,
                                    A => A,
                                    B => B,
                                    Cin => Cin,
                                    Sum => Sum);


stimuli: PROCESS
 begin
    A <= "0000";
    B <= "0000";
    WAIT FOR T;

    FOR j IN 1 TO 15 LOOP
        FOR i IN 1 TO 15 LOOP
            A <= A + 1;
            WAIT FOR T;
        end LOOP;
        B <= B + 1;
        WAIT FOR T;
    END LOOP;

   IF Cin = '0' THEN
    Cin <= '1';
   ELSE
    Cin <= '0';
   END IF;

 end PROCESS;

  clk_gen: process begin
    clk <= '0';
    wait for T/2;
    clk <= '1';
    wait for T/2;
 end process;



end Bench;
```

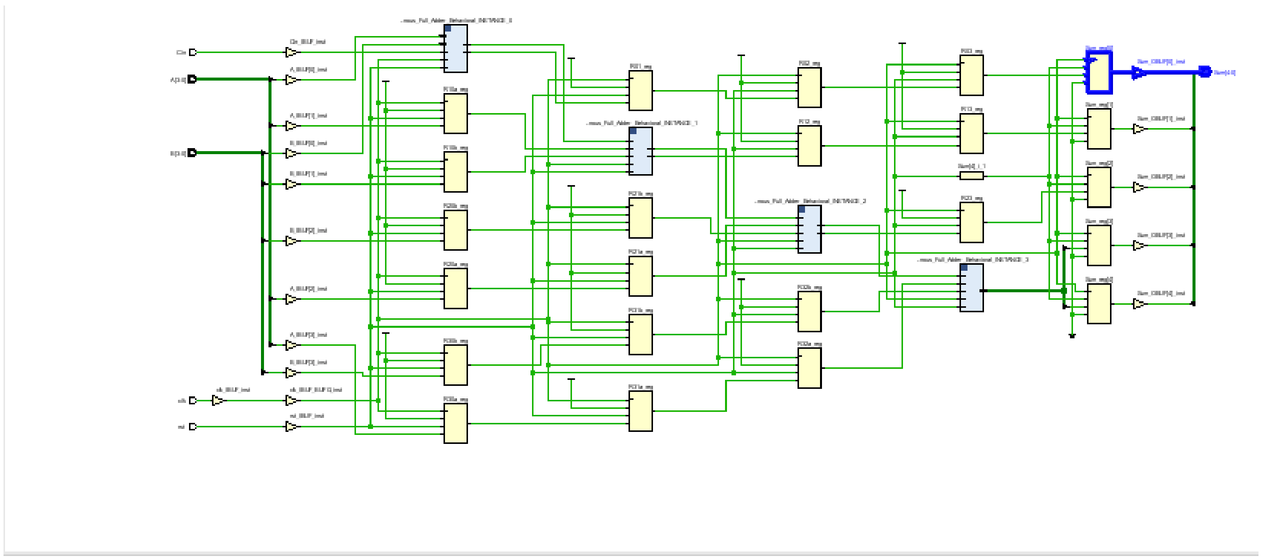


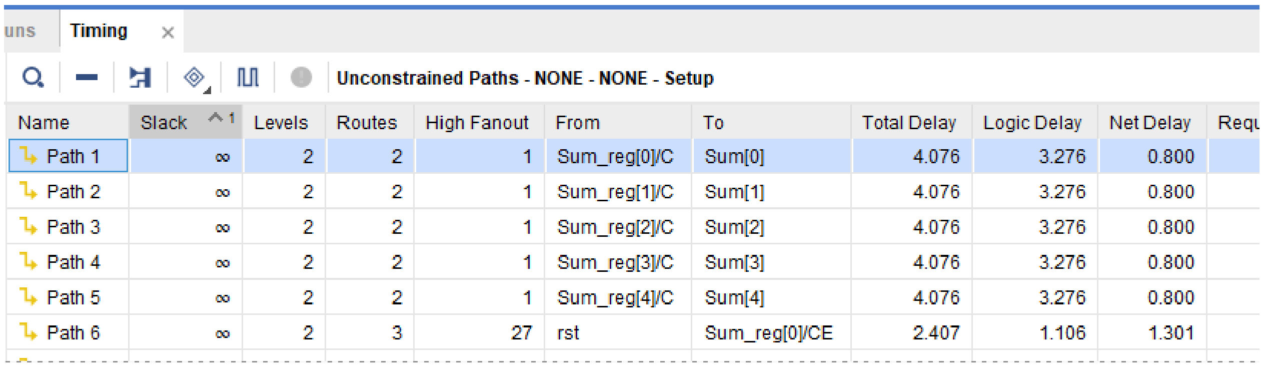






Critical Path και Συνολική Καθυστέρηση





uns | Timing ×

Unconstrained Paths - NONE - NONE - Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requ
Path 1	∞	2	2	1	Sum_reg[0]/C	Sum[0]	4.076	3.276	0.800	
Path 2	∞	2	2	1	Sum_reg[1]/C	Sum[1]	4.076	3.276	0.800	
Path 3	∞	2	2	1	Sum_reg[2]/C	Sum[2]	4.076	3.276	0.800	
Path 4	∞	2	2	1	Sum_reg[3]/C	Sum[3]	4.076	3.276	0.800	
Path 5	∞	2	2	1	Sum_reg[4]/C	Sum[4]	4.076	3.276	0.800	
Path 6	∞	2	3	27	rst	Sum_reg[0]/CE	2.407	1.106	1.301	

Όπως φαίνεται από την παραπάνω εικόνα, το critical path είναι από τον τελευταίο καταχωρητή στο Sum[0] και η συνολική καθυστέρηση είναι 4.076 ns.

Ο Παράλληλος Αθροιστής της 2ης Εργαστηριακής Άσκησης είχε ως καθυστέρηση, και άρα ως κύκλο ρολογιού θεωρώντας ότι είναι σύγχρονος, 5.942ns. Άρα, με την μέθοδο pipeline μειώσαμε την καθυστέρηση στο 1/3 της αρχικής. Όσον, αφορά την κατανάλωση πόρων, η μέθοδος pipeline χρησιμοποιεί σαφώς περισσότερο υλικό. Πιο συγκεκριμένα χρησιμοποιεί 3 επιπλέον καταχωρητές σε κάθε στάδιο, σε αντίθεση με τον προηγούμενο που δεν χρησιμοποιεί κανέναν. Όμως, τα πλεονεκτήματα του pipelining ξεπερνούν το παραπάνω μειονέκτημα καθώς το σύστημά μας γίνεται αρκετά πιο γρήγορο και καταναλώνει λιγότερη ισχύ.

3. Συστολικός Πολλαπλασιαστής διάδοσης κρατουμένου 4ων bit

Κώδικας και δομικό διάγραμμα (RTL)

D Flip Flop

```
Library IEEE;
USE IEEE.Std_logic_1164.all;

entity D_Flip_Flop_behavioral is
   port(
      Q : out std_logic;
      Clk : in std_logic;
      rst : in std_logic;
      D :in  std_logic
   );
end D_Flip_Flop_behavioral;
architecture Behavioral of D_Flip_Flop_behavioral is
begin
 process(Clk)
 begin
    if(rst='1') then
        Q<='0';
    elsif(rising_edge(Clk)) then
        Q <= D;
    end if;
 end process;
end Behavioral;
```

Building Block

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

--The following is the implementation of the Building block that we will use for
↪  the Multiplication design.
--We can observe the diagram of the building block at the upper left corner of
↪  the Multiplication diagram we quoted below.
entity Building_Block is
  port(
       clk : in std_logic;
       rst : in std_logic;
       Sin : in std_logic;
       Ain   : in std_logic;
       Bin   : in std_logic;
       Cin : in std_logic;
       Sout: out std_logic;
       Cout: out std_logic;
       Aout: out std_logic;
       Bout: out std_logic
      );
end entity; -- Building Block

architecture structural_arch of Building_Block is

  -------------------------------------------
  -- Declarations of lower level components
  -- used in this level of hierarchy
  component Sychronous_Full_Adder_Behavioral is
    port(
       clk : in std_logic;
       rst : in std_logic;
       A : in  std_logic;
       B   : in  std_logic;
       Cin : in std_logic;
       Sum   : out  std_logic;
       Cout    : out  std_logic
        );
  end component;

  component D_Flip_Flop_behavioral is
    port(
      Q : out std_logic;
      Clk : in std_logic;
      rst : in std_logic;
      D :in  std_logic
        );
  end component;
  -------------------------------------
  -- Declarations of internal signals
  -- used in this level of hierarchy
signal C1,C3 : std_logic;  -- || C1 ->  input FA || C3 -> between D's ||
begin
  C1<=Ain AND Bin;
  -------------------------------------
  -- LEVEL 0 component instantiation --
  -------------------------------------
  Sychronous_Full_Adder_Behavioral_INSTANCE_0 : Sychronous_Full_Adder_Behavioral
    port map (
            clk => clk,
            rst => rst,
            A   => C1,  --result of AND
            B   => Sin, --result of above addition
            Cin => Cin,
            Sum =>  Sout,
            Cout =>  Cout
        );
  D_Flip_Flop_behavioral_INSTANCE_0 : D_Flip_Flop_behavioral
    port map (
      Q => Bout,
      Clk => clk,
      rst => rst,
      D => Bin
       );
  -------------------------------------
  -- LEVEL 1 component instantiation --
  -------------------------------------
  D_Flip_Flop_behavioral_INSTANCE_1 : D_Flip_Flop_behavioral
    port map (
      Q => C3,
      Clk => clk,
      rst => rst,
      D => Ain
       );
  -------------------------------------
  -- LEVEL 2 component instantiation --
  -------------------------------------
  D_Flip_Flop_behavioral_INSTANCE_2 : D_Flip_Flop_behavioral
    port map (
      Q => Aout,
      Clk => clk,
      rst => rst,
      D => C3
       );

end architecture ; -- arch
```

Σημειώνεται ότι το Building Block φαίνεται στην άνω δεξιά γωνία της φωτογραφίας που ακολουθεί και που δείχνει την διαγραμματική υλοποίηση του πολλαπλασιαστή.

Πολλαπλασιαστής

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

--Our design is based on the diagram given below the code part.
--Regarding the delays we use code as the one given below, to create delays via
↪  the use of flip - flops(D):
--
--  delay_n:process (clk)                 --n clock cycles
--  begin
--    if rising_edge(clk) then
```

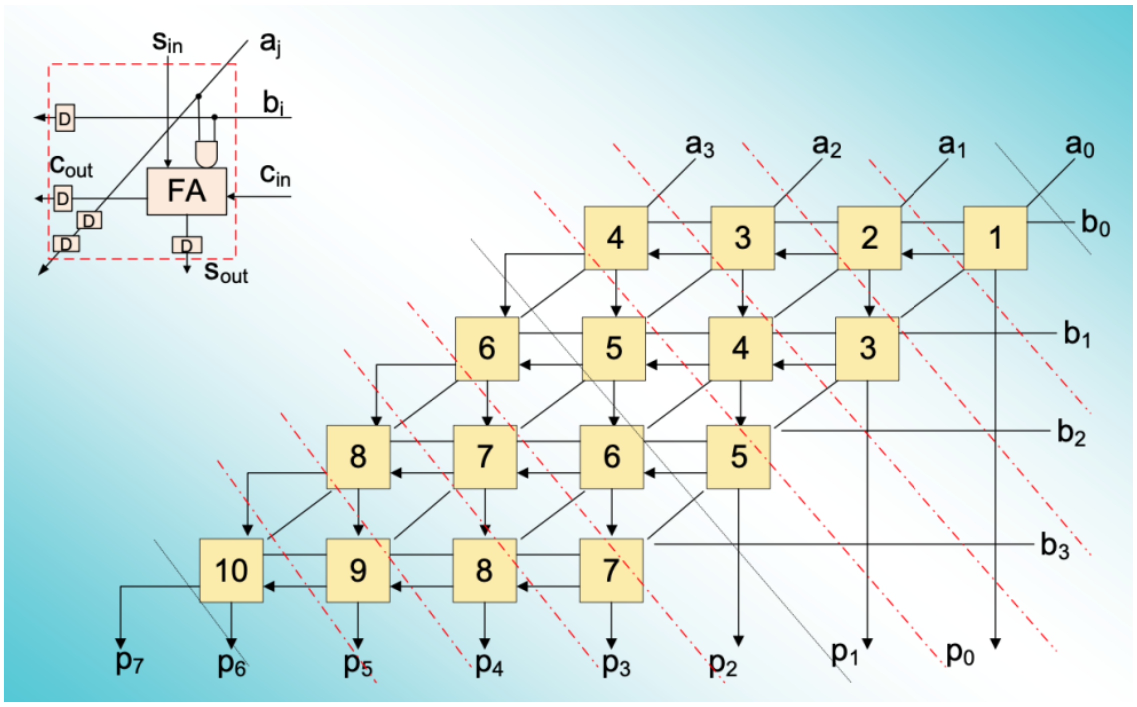
```vhdl
--      delay0 <= delay1;
--      ....
--      ....
--      ....
--      delayn-1 <= A(x);
--    end if;
--  end process;

entity Sychronous_Systolic_4bit_Multiplier is
    Port ( clk : in std_logic;
           rst : in std_logic;
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Cin : in STD_LOGIC;
           Product : out STD_LOGIC_VECTOR (7 downto 0));
end Sychronous_Systolic_4bit_Multiplier;

architecture Structural of Sychronous_Systolic_4bit_Multiplier is
    --Our single compoment used 16 times is the Sychronous FA
    component Building_Block is
    port(
        clk : in std_logic;
        rst : in std_logic;
        Sin : in std_logic;
        Ain : in std_logic;
        Bin : in std_logic;
        Cin : in std_logic;
        Sout: out std_logic;
        Cout: out std_logic;
        Aout: out std_logic;
        Bout: out std_logic
         );
  end component;
  --About Delays:
  --We have 4 categories:
  --1)Delays for the Cin of each FA
  --2)Delays for the "straight"(according to the diagram) inputs ,B
  --3)Delays for the "diagonal"(according to the diagram) inputs ,A
  --4)Delays for the Outputs of the last Full Adders-counting from top to
  ↪  bottom-(giving us the multiplication result)
  --
  --In the following code we distinguish every type of delay from each other
  ↪  according to the above enumeration.


  --Signals regarding subtotals
  signal P0 : std_logic_vector(9 downto 0):="0000000000"; --ready
  signal P1 : std_logic_vector(8 downto 0):="000000000"; --ready
  signal P2 : std_logic_vector(7 downto 0):="00000000"; --ready
  signal P3 : std_logic_vector(6 downto 0):="0000000"; --ready
  signal P4 : std_logic_vector(4 downto 0):="00000"; --ready
  signal P5 : std_logic_vector(2 downto 0):="000"; --ready
  signal P6 : std_logic:='0'; --ready
  signal P7 : std_logic:='0'; --ready

  --signals regarding internal transports
  signal C0 : std_logic_vector(3 downto 0):="0000"; --Cout each gate
  signal C1 : std_logic_vector(3 downto 0):="0000"; --Cout each gate
  signal C2 : std_logic_vector(3 downto 0):="0000"; --Cout each gate
  signal C3 : std_logic_vector(3 downto 0):="0000"; --Cout each gate
  signal A0 : std_logic_vector(3 downto 0):="0000"; --diagonal
  signal A1 : std_logic_vector(3 downto 0):="0000"; --diagonal
  signal A2 : std_logic_vector(3 downto 0):="0000"; --diagonal
  signal A3 : std_logic_vector(3 downto 0):="0000"; --diagonal
  signal B0 : std_logic_vector(3 downto 0):="0000"; --straight
  signal B1 : std_logic_vector(3 downto 0):="0000"; --straight
  signal B2 : std_logic_vector(3 downto 0):="0000"; --straight
  signal B3 : std_logic_vector(3 downto 0):="0000"; --straight

  --delays for A
  signal delayA1 : std_logic:='0';
  signal delayA2,delayA3 : std_logic:='0';
  signal delayA4,delayA5,delayA6 : std_logic:='0';

  --delays for B
  signal delayB1,delayB2 : std_logic:='0';
  signal delayB3,delayB4 : std_logic:='0';
  signal delayB5,delayB6,delayB7 : std_logic:='0';
  signal delayB8,delayB9,delayB10 : std_logic:='0';
  signal delayB11,delayB12,delayB13,delayB14 : std_logic:='0';
  --delays for C
  signal delayC1,delayC3 : std_logic:='0';
  signal delayC2,delayC4 : std_logic:='0';
  signal delayC5,delayC6 : std_logic:='0';
begin
--
--
--
--
--
--
--

  Building_Block_INSTANCE_0: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => '0',
               Ain => A(0),
               Bin => B(0),
               Cin => '0',
               Sout => P0(0),
               Cout => C0(0),
               Aout => A0(0),
               Bout => B0(0));

--Type 4
 delay_P0:process (clk)                --10 clock cycles
begin
  if rising_edge(clk) then
    Product(0) <= P0(9);
    P0(9) <= P0(8);
    P0(8) <= P0(7);
    P0(7) <= P0(6);
    P0(6) <= P0(5);
    P0(5) <= P0(4);
    P0(4) <= P0(3);
    P0(3) <= P0(2);
    P0(2) <= P0(1);
    P0(1) <= P0(0);
  end if;
end process;
--Type 3
 delay_1:process (clk)
begin
  if rising_edge(clk) then
    delayA1 <= A(1);
  end if;
end process;

  Building_Block_INSTANCE_1: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => '0',
               Ain => delayA1,
               Bin => B0(0),
               Cin => C0(0),
               Sout =>P1(0),
               Cout => C0(1),
               Aout => A1(0),
               Bout => B0(1));
--Type 3
 delay_2:process (clk)                --2 clock cycles
begin
  if rising_edge(clk) then
```

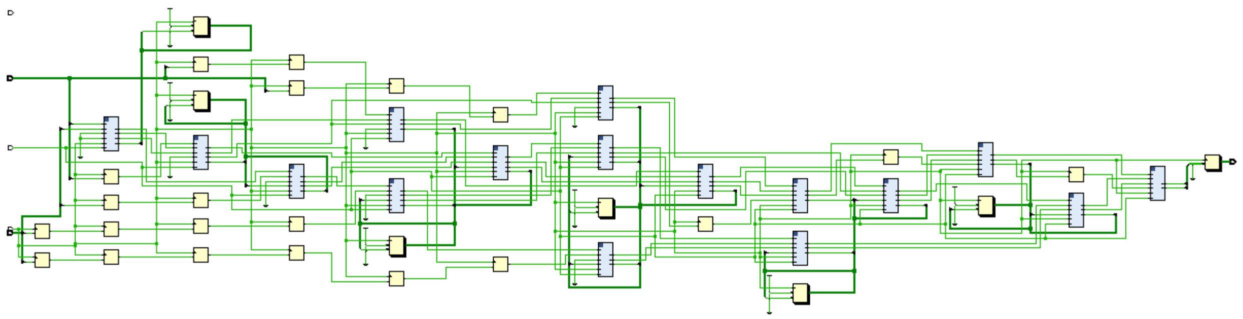
```
    delayA2 <= delayA3;
    delayA3 <= A(2);
  end if;
end process;

  Building_Block_INSTANCE_2: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => '0',
               Ain => delayA2,
               Bin => B0(1),
               Cin => C0(1),
               Sout =>P2(0),
               Cout => C0(2),
               Aout => A2(0),
               Bout => B0(2));
--Type 3
 delay_3:process (clk)                --3 clock cycles
begin
  if rising_edge(clk) then
    delayA4 <= delayA5;
    delayA5 <= delayA6;
    delayA6 <= A(3);
  end if;
end process;

  Building_Block_INSTANCE_3: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => '0',
               Ain => delayA4,
               Bin => B0(2),
               Cin => C0(1),
               Sout =>P3(0),
               Cout => C0(3),
               Aout => A3(0),
               Bout => B0(3));
--Type 2
 delay_4:process (clk)                --2 clock cycles
begin
  if rising_edge(clk) then
    delayB1 <= delayB2;
    delayB2 <= B(1);
  end if;
end process;

  Building_Block_INSTANCE_4: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P1(0),
               Ain => A0(0),
               Bin => delayB1,
               Cin => '0',
               Sout =>P1(1),
               Cout => C1(0),
               Aout => A0(1),
               Bout => B1(0));

--Type 4
 delay_P1:process (clk)                --8 clock cycles
begin
  if rising_edge(clk) then
    Product(1) <= P1(8);
    P1(8) <= P1(7);
    P1(7) <= P1(6);
    P1(6) <= P1(5);
    P1(5) <= P1(4);
    P1(4) <= P1(3);
    P1(3) <= P1(2);
    P1(2) <= P1(1);
  end if;
end process;

  Building_Block_INSTANCE_5: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P2(0),
               Ain => A1(0),
               Bin => B1(0),
               Cin => C1(0),
               Sout =>P2(1),
               Cout => C1(1),
               Aout => A1(1),
               Bout => B1(1));

  Building_Block_INSTANCE_6: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P3(0),
               Ain => A2(0),
               Bin => B1(1),
               Cin => C1(1),
               Sout =>P3(1),
               Cout => C1(2),
               Aout => A2(1),
               Bout => B1(2));
```

```vhdl
--Type 1
 delay_5:process (clk)                  --5 clock cycles
begin
  if rising_edge(clk) then
    delayC1 <= C0(3);
    --delayC2 <= C0(3);
  end if;
end process;

  Building_Block_INSTANCE_7: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => delayC1,
               Ain => A3(0),
               Bin => B1(2),
               Cin => C1(2),
               Sout =>P4(0),
               Cout => C1(3),
               Aout => A3(1),
               Bout => B1(3));
--Type 2
 delay_6:process (clk)                  --5 clock cycles
begin
  if rising_edge(clk) then
    delayB3 <= delayB4;
    delayB4 <= delayB5;
    delayB5 <= delayB6;
    delayB6 <= B(2);
  end if;
end process;

  Building_Block_INSTANCE_8: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P2(1),
               Ain => A0(1),
               Bin => delayB3,
               Cin => '0',
               Sout =>P2(2),
               Cout => C2(0),
               Aout => A0(2),
               Bout => B2(0));
--Type 4
 delay_P2:process (clk)                  --7 clock cycles
begin
  if rising_edge(clk) then
    Product(2) <= P2(7);
    P2(7) <= P2(6);
    P2(6) <= P2(5);
    P2(5) <= P2(4);
    P2(4) <= P2(3);
    P2(3) <= P2(2);
  end if;
end process;

  Building_Block_INSTANCE_9: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P3(1),
               Ain => A1(1),
               Bin => B2(0),
               Cin => C2(0),
               Sout =>P3(2),
               Cout => C2(1),
               Aout => A1(2),
               Bout => B2(1));

  Building_Block_INSTANCE_10: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P4(0),
               Ain => A2(1),
               Bin => B2(1),
               Cin => C2(1),
               Sout =>P4(1),
               Cout => C2(2),
               Aout => A2(2),
               Bout => B2(2));
--Type 1
 delay_7:process (clk)                 --2 clock cycles
begin
  if rising_edge(clk) then
    delayC3 <= C1(3);
    --delayC4 <= C1(3);
  end if;
end process;

  Building_Block_INSTANCE_11: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => delayC3,
               Ain => A3(1),
               Bin => B2(2),
               Cin => C2(2),
               Sout =>P5(0),
               Cout => C2(3),
                Aout => A3(2),
                Bout => B2(3));
--Type 2
 delay_8:process (clk)                  --6 clock cycles
begin
  if rising_edge(clk) then
    delayB7 <= delayB8;
    delayB8 <= delayB9;
    delayB9 <= delayB10;
    delayB10 <= delayB11;
    delayB11 <= delayB12;
    delayB12 <= B(3);
  end if;
end process;

  Building_Block_INSTANCE_12: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P3(2),
               Ain => A0(2),
               Bin => delayB7,
               Cin => '0',
               Sout =>P3(3),
               Cout => C3(0),
               Aout => A0(3),
               Bout => B3(0));
--Type 4
 delay_P3:process (clk)                  --4 clock cycles
begin
  if rising_edge(clk) then
    Product(3) <= P3(6);
    P3(6) <= P3(5);
    P3(5) <= P3(4);
    P3(4) <= P3(3);
  end if;
end process;

  Building_Block_INSTANCE_13: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P4(1),
               Ain => A1(2),
               Bin => B3(0),
               Cin => C3(0),
               Sout =>P4(2),
               Cout => C3(1),
               Aout => A1(3),
               Bout => B3(1));
```

```vhdl
--Type 4
 delay_P4:process (clk)                --3 clock cycles
begin
  if rising_edge(clk) then
    Product(4) <= P4(4);
    P4(4) <= P4(3);
    P4(3) <= P4(2);
  end if;
end process;

  Building_Block_INSTANCE_14: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => P5(0),
               Ain => A2(2),
               Bin => B3(1),
               Cin => C3(1),
               Sout =>P5(1),
               Cout => C3(2),
               Aout => A2(3),
               Bout => B3(2));
--Type 4
 delay_P5:process (clk)                --2 clock cycles
begin
  if rising_edge(clk) then
    Product(5) <= P5(2);
    P5(2) <= P5(1);
  end if;
end process;
--Type 1
 delay_9:process (clk)                --2 clock cycles
begin
  if rising_edge(clk) then
    delayC5 <= C2(3);
    --delayC6 <= C2(3);
  end if;
end process;

  Building_Block_INSTANCE_15: Building_Block
    port map ( clk => clk,
               rst => rst,
               Sin => delayC5,
               Ain => A3(2),
               Bin => B3(2),
               Cin => C3(2),
               Sout =>P6,
               Cout => P7,
               Aout => A3(3),
                  Bout => B3(3));
--Type 4
 delay_P6_P7:process (clk)                    --2 clock cycles
begin
  if rising_edge(clk) then
    Product(7) <= P7;
    Product(6) <= P6;
  end if;
end process;

end Structural;
```

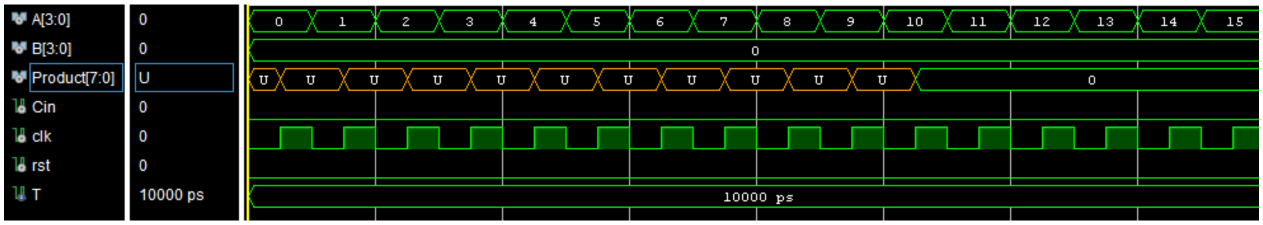
Παρακάτω δίνεται το διάγραμμα το οποίο παρουσιάζει οπτικά την υλοποίηση που πραγματοποιήσαμε καθώς και το rtl σχηματικό.

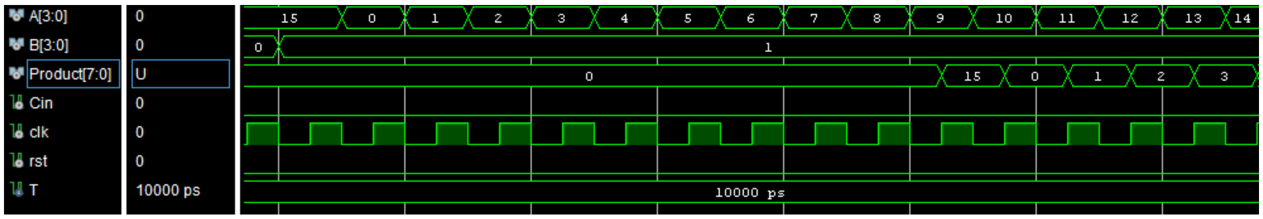


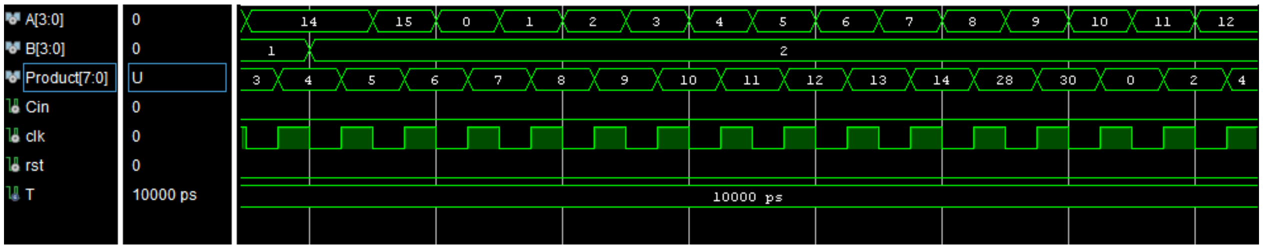


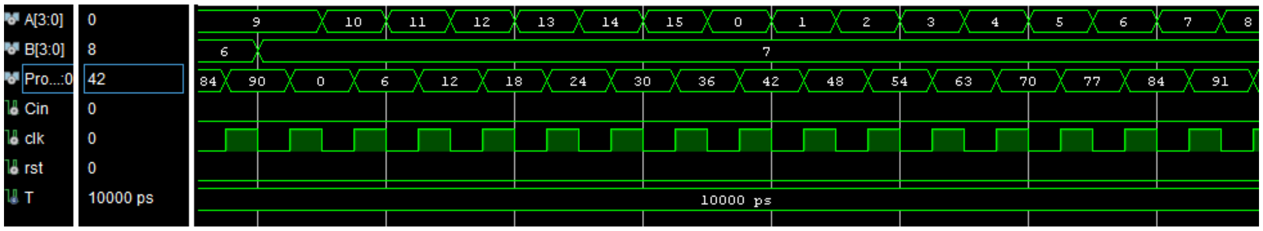
Κώδικας testbench και Κυματομορφή εξόδου

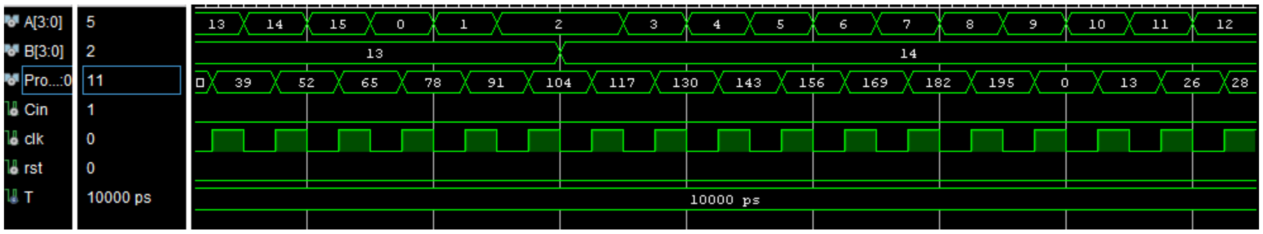
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.std_logic_unsigned.ALL;


entity Sychronous_Systolic_4bit_Multiplier_tb is
--  Port ( );
end entity;

architecture Bench of Sychronous_Systolic_4bit_Multiplier_tb is

COMPONENT Sychronous_Systolic_4bit_Multiplier is
    Port ( clk : in std_logic;
           rst : in std_logic;
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Cin : in STD_LOGIC;
           Product : out STD_LOGIC_VECTOR (7 downto 0));
end COMPONENT;

SIGNAL A, B : STD_LOGIC_VECTOR(3 downto 0);
SIGNAL Product : STD_LOGIC_VECTOR(7 downto 0);
SIGNAL Cin : STD_LOGIC := '0';
SIGNAL clk :  std_logic;
SIGNAL rst :  std_logic := '0';

CONSTANT T : TIME := 10 ns;

begin

uut: Sychronous_Systolic_4bit_Multiplier PORT MAP (
                                    clk =>clk,
                                    rst => rst,
                                    A => A,
                                    B => B,
                                    Cin => Cin,
                                    Product => Product);


stimuli: PROCESS
 begin
    A <= "0000";
    B <= "0000";
    WAIT FOR T;

    FOR j IN 1 TO 15 LOOP
        FOR i IN 1 TO 15 LOOP
            A <= A + 1;
            WAIT FOR T;
        end LOOP;
        B <= B + 1;
        WAIT FOR T;
    END LOOP;

   IF Cin = '0' THEN
    Cin <= '1';
   ELSE
    Cin <= '0';
   END IF;

 end PROCESS;

  clk_gen: process begin
    clk <= '0';
    wait for T/2;
    clk <= '1';
    wait for T/2;
 end process;



end Bench;
```


A[3:0] 0 | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
B[3:0] 0 | 0
Product[7:0] U | U U U U U U U U U U U 0
Cin 0
clk 0
rst 0
T 10000 ps | 10000 ps

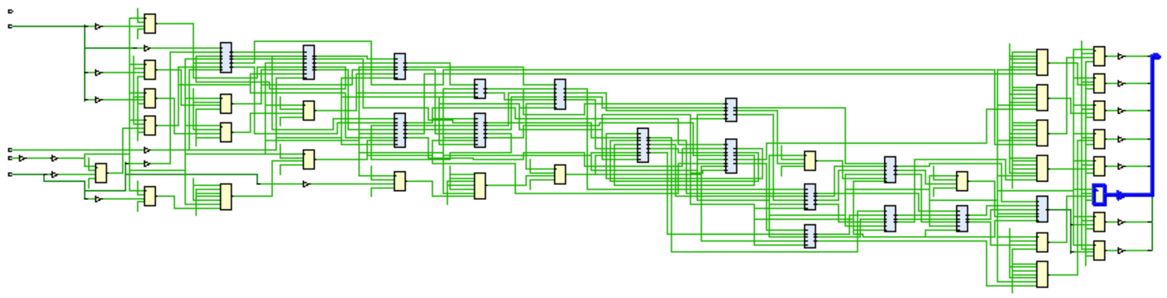

A[3:0] 0 | 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
B[3:0] 0 | 0 1
Product[7:0] U | 0 15 0 1 2 3
Cin 0
clk 0
rst 0
T 10000 ps | 10000 ps

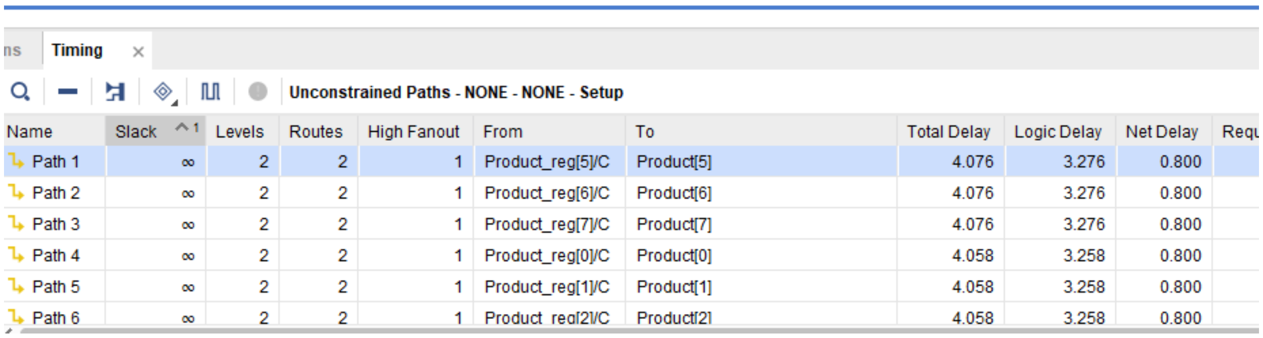

A[3:0] 0 | 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12
B[3:0] 0 | 1 2
Product[7:0] U | 3 4 5 6 7 8 9 10 11 12 13 14 28 30 0 2 4
Cin 0
clk 0
rst 0
T 10000 ps | 10000 ps





Critical Path και Συνολική Καθυστέρηση





Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requ
Path 1	∞	2	2	1	Product_reg[5]/C	Product[5]	4.076	3.276	0.800	
Path 2	∞	2	2	1	Product_reg[6]/C	Product[6]	4.076	3.276	0.800	
Path 3	∞	2	2	1	Product_reg[7]/C	Product[7]	4.076	3.276	0.800	
Path 4	∞	2	2	1	Product_reg[0]/C	Product[0]	4.058	3.258	0.800	
Path 5	∞	2	2	1	Product_reg[1]/C	Product[1]	4.058	3.258	0.800	
Path 6	∞	2	2	1	Product_reg[2]/C	Product[2]	4.058	3.258	0.800	

Όπως φαίνεται από την παραπάνω εικόνα, το critical path είναι από τον τελευταίο καταχωρητή στο Product[5] και η συνολική καθυστέρηση είναι 4.076ns.