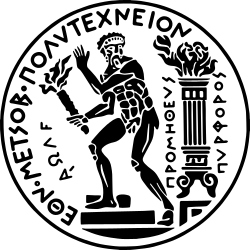
**Εθνικό Μετσόβιο Πολυτεχνείο** 

**Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών**

**Υπολογιστών**

Εξάμηνο: 8ο Ακ. Έτος: 2024- 2025

# «Ψηφιακά Συστήματα VLSI»

Χαράλαμπος Παπαδόπουλος (ΑΜ: 03120199)

Νικόλαος Παπακωνσταντόπουλος (ΑΜ: 03120069)

Ομάδα: 45

# Σχεδίαση Μονάδων Υλικού με την Τεχνική Pipelining

Ο σκοπός της παρούσας άσκησης είναι η υλοποίηση μονάδων υλικού (αθροιστή και συστολικού πολλαπλασιαστή) με χρήση της μεθόδου Pipelining. Σε αυτή την τεχνική, ο υπολογισμός του αποτελέσματος προκύπτει από συγκεκριμένο αριθμό σταδίων καθένα από τα οποία “πυροδοτείται” με μια ανερχόμενη ακμή ρολογιού. Συνεπώς, το επιθυμητό αποτέλεσμα παράγεται με συγκεκριμένη καθυστέρηση στην έξοδο, (σύνολο σταδίων του pipeline επί την περίοδο ρολογιού).

Προς τούτο, πρέπει να είναι σύγχρονες οι δομικές μονάδες των ψηφιακών κυκλωμάτων (σύγχρονες σημαίνει ότι ενημερώνουν το αποτέλεσμα σε κάθε ανερχόμενη ακμή ρολογιού). Παρατίθενται λοιπόν οι δομικές μονάδες του σύγχρονου αθροιστή, του D Flip Flop και του pipeline cell που χρησιμοποιείται για τον πολλαπλασιαστή.

### Synchronous Adder

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity sync\_adder is

Port (A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

rst : in std\_logic;

clk : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end sync\_adder;

architecture Behavioral of sync\_adder is

begin

process(clk, rst)

begin

if rst = '1' then

Sum <= '0';

Cout <= '0';

elsif rising\_edge(clk) then

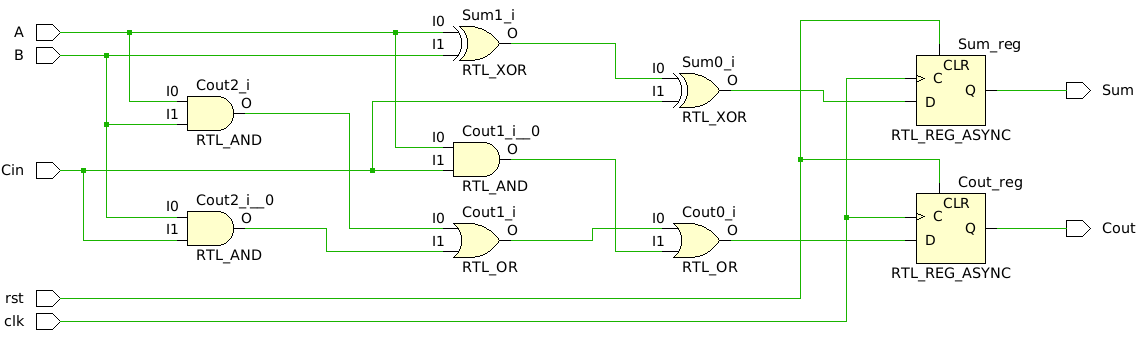
Sum <= A xor B xor Cin;

Cout <= (A and B) or (B and Cin) or (A and Cin);

end if;

end process;

end Behavioral;



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity sync\_adder\_tb is

end sync\_adder\_tb;

architecture Behavioral of sync\_adder\_tb is

component sync\_adder is

Port (A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

rst : in std\_logic;

clk : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end component;

signal A, B, Cin, Sum, Cout : std\_logic;

signal rst, clk : std\_logic := '0';

begin

uut: sync\_adder port map (

A => A,

B => B,

Cin => Cin,

rst => rst,

clk => clk,

Sum => Sum,

Cout => Cout

);

clk\_proc: process begin

clk <= '0';

wait for 5 ns;

clk <= '1';

wait for 5 ns;

end process;

stim\_proc: process begin

rst <= '1';

A <= '1';

B <= '1';

Cin <= '1';

wait for 10 ns;

rst <= '0';

A <= '1';

B <= '0';

Cin <= '0';

wait for 10 ns;

A <= '1';

B <= '1';

Cin <= '0';

wait for 10 ns;

A <= '1';

B <= '1';

Cin <= '1';

wait for 10 ns;

A <= '0';

B <= '0';

Cin <= '0';

wait for 10 ns;

A <= '0';

B <= '0';

Cin <= '1';

wait for 10 ns;

A <= '0';

B <= '1';

Cin <= '1';

wait for 10 ns;

A <= '1';

B <= '1';

Cin <= '1';

wait for 10 ns;

A <= '1';

B <= '0';

Cin <= '1';

wait for 10 ns;

A <= '0';

B <= '0';

Cin <= '0';

wait for 10 ns;

A <= '1';

B <= '1';

Cin <= '1';

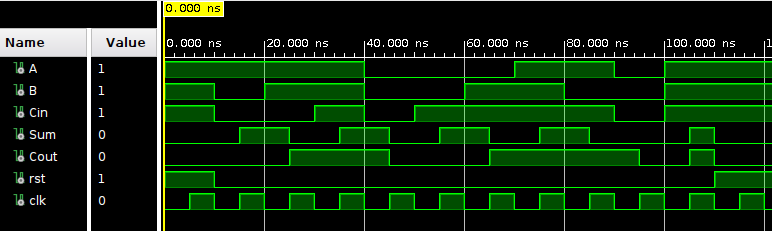
wait for 10 ns;

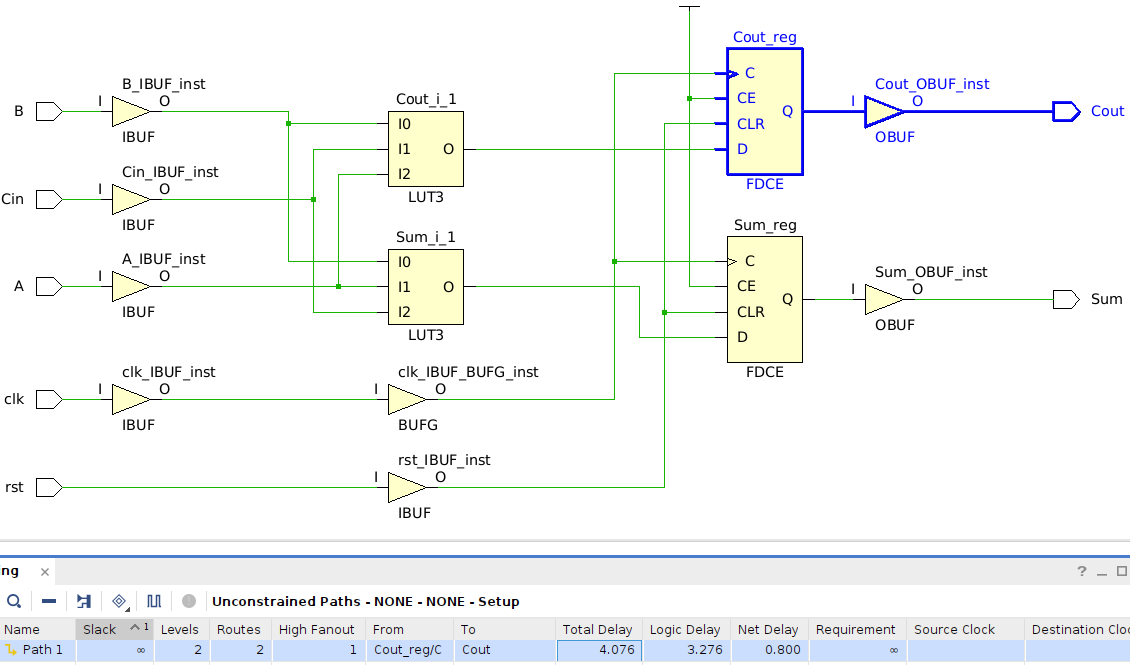
rst <= '1';

wait;

end process;

end Behavioral;





*Critical Path (D flip flop -> cout): 4.076 ns*

### D Flip Flop

Library IEEE;

USE IEEE.Std\_logic\_1164.all;

entity D\_flip\_flop is

port(

clk :in std\_logic;

rst: in std\_logic;

D :in std\_logic;

Q : out std\_logic

);

end D\_flip\_flop;

architecture Behavioral of D\_flip\_flop is

begin

process(clk,rst)

begin

if(rst='1') then

Q <= '0';

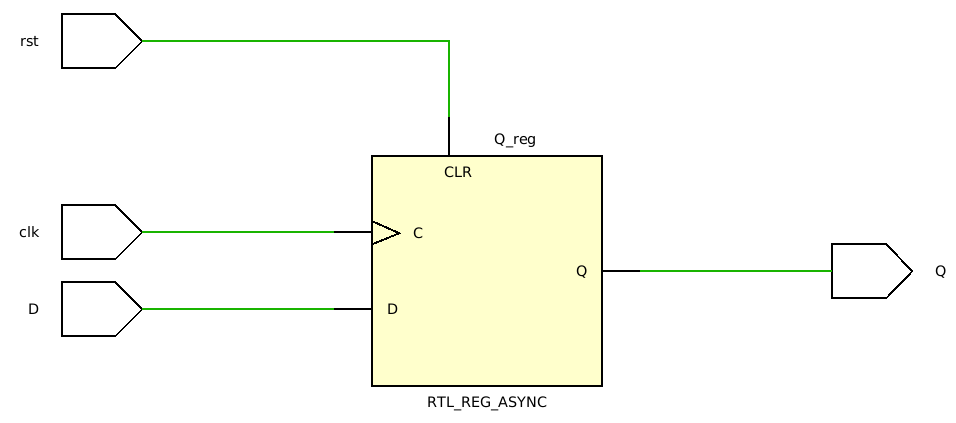
elsif(rising\_edge(clk)) then

Q <= D;

end if;

end process;

end Behavioral;



## Pipeline Adder

Pipeline adder

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity pipeline\_adder is

Port (

A : in std\_logic\_vector(3 downto 0);

B : in std\_logic\_vector(3 downto 0);

cin : in std\_logic;

cout : out std\_logic;

sum : out std\_logic\_vector(3 downto 0);

clk : in std\_logic;

rst : in std\_logic

);

end pipeline\_adder;

architecture Structural of pipeline\_adder is

component sync\_adder is

Port (A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

rst : in std\_logic;

clk : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end component;

component D\_flip\_flop is

port(

clk :in std\_logic;

rst: in std\_logic;

D :in std\_logic;

Q : out std\_logic

);

end component;

signal stage1\_cout, stage1\_sum, stage2\_cout, stage2\_sum, stage3\_cout, stage3\_sum : std\_logic;

signal stage1\_D\_A1, stage1\_D\_B1 : std\_logic;

signal stage1\_D\_A2, stage1\_D\_B2, stage2\_D\_A2, stage2\_D\_B2 : std\_logic;

signal stage1\_D\_A3, stage1\_D\_B3, stage2\_D\_A3, stage2\_D\_B3, stage3\_D\_A3, stage3\_D\_B3 : std\_logic;

signal stage2\_D0\_out, stage3\_D0\_out : std\_logic;

signal stage3\_D1\_out : std\_logic;

begin

-- stage 1

stage1\_FA: sync\_adder port map(A => A(0), B => B(0), cin => cin, rst => rst, clk => clk, Sum => stage1\_sum, Cout => stage1\_cout);

stage1\_DFF\_A1: D\_Flip\_Flop port map(clk => clk, rst => rst, D => A(1), Q => stage1\_D\_A1);

stage1\_DFF\_B1: D\_Flip\_Flop port map(clk => clk, rst => rst, D => B(1), Q => stage1\_D\_B1);

stage1\_DFF\_A2: D\_Flip\_Flop port map(clk => clk, rst => rst, D => A(2), Q => stage1\_D\_A2);

stage1\_DFF\_B2: D\_Flip\_Flop port map(clk => clk, rst => rst, D => B(2), Q => stage1\_D\_B2);

stage1\_DFF\_A3: D\_Flip\_Flop port map(clk => clk, rst => rst, D => A(3), Q => stage1\_D\_A3);

stage1\_DFF\_B3: D\_Flip\_Flop port map(clk => clk, rst => rst, D => B(3), Q => stage1\_D\_B3);

-- stage 2

stage2\_DFF\_S0: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage1\_sum, Q => stage2\_D0\_out);

stage2\_FA: sync\_adder port map(A => stage1\_D\_A1, B => stage1\_D\_B1, cin => stage1\_cout, rst => rst, clk => clk, Sum => stage2\_sum, Cout => stage2\_cout);

stage2\_DFF\_A2: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage1\_D\_A2, Q => stage2\_D\_A2);

stage2\_DFF\_B2: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage1\_D\_B2, Q => stage2\_D\_B2);

stage2\_DFF\_A3: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage1\_D\_A3, Q => stage2\_D\_A3);

stage2\_DFF\_B3: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage1\_D\_B3, Q => stage2\_D\_B3);

-- stage 3

stage3\_DFF\_S0: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage2\_D0\_out, Q => stage3\_D0\_out);

stage3\_DFF\_S1: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage2\_sum, Q => stage3\_D1\_out);

stage3\_FA: sync\_adder port map(A => stage2\_D\_A2, B => stage2\_D\_B2, cin => stage2\_cout, rst => rst, clk => clk, Sum => stage3\_sum, Cout => stage3\_cout);

stage3\_DFF\_A3: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage2\_D\_A3, Q => stage3\_D\_A3);

stage3\_DFF\_B3: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage2\_D\_B3, Q => stage3\_D\_B3);

-- stage 4

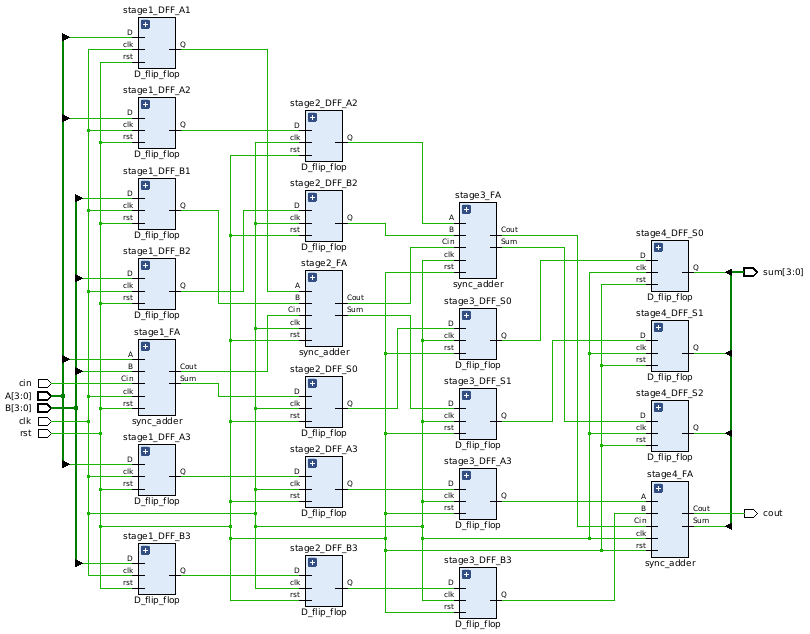
stage4\_DFF\_S0: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage3\_D0\_out, Q => sum(0));

stage4\_DFF\_S1: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage3\_D1\_out, Q => sum(1));

stage4\_DFF\_S2: D\_Flip\_Flop port map(clk => clk, rst => rst, D => stage3\_sum, Q => sum(2));

stage4\_FA: sync\_adder port map(A => stage3\_D\_A3, B => stage3\_D\_B3, cin => stage3\_cout, rst => rst, clk => clk, Sum => sum(3), Cout => cout);

end Structural;



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use STD.TEXTIO.ALL;

entity pipeline\_adder\_tb is

end pipeline\_adder\_tb;

architecture testbench of pipeline\_adder\_tb is

component pipeline\_adder is

Port (

A : in std\_logic\_vector(3 downto 0);

B : in std\_logic\_vector(3 downto 0);

cin : in std\_logic;

cout : out std\_logic;

sum : out std\_logic\_vector(3 downto 0);

clk : in std\_logic;

rst : in std\_logic

);

end component;

signal A, B : std\_logic\_vector(3 downto 0);

signal cin, cout : std\_logic;

signal sum : std\_logic\_vector(3 downto 0);

signal clk, rst : std\_logic;

constant CLOCK\_PERIOD : time := 10ns;

begin

UUT: pipeline\_adder port map(

A => A,

B => B,

cin => cin,

cout => cout,

sum => sum,

clk => clk,

rst => rst

);

GEN\_CLK : process

begin

clk <= '0';

wait for CLOCK\_PERIOD/2;

clk <= '1';

wait for CLOCK\_PERIOD/2;

end process;

STIMULUS: process

begin

-- use rst = 1 to reset the pipeline\_adder

-- use rst = 0 to enable the pipeline\_adder

rst <= '1';

wait for CLOCK\_PERIOD;

rst <= '0';

wait for CLOCK\_PERIOD;

-- now 2 periods have passed, so we can start testing

-- test case 1: 5 + 10 = 15

A <= "0101"; B <= "1010"; cin <= '0';

wait for CLOCK\_PERIOD;

-- test case 2: 3 + 7 + 1 = 11

A <= "0011"; B <= "0111"; cin <= '1';

wait for CLOCK\_PERIOD;

-- test case 3: 12 + 15 = 27 (11)

A <= "1100"; B <= "1111"; cin <= '0';

wait for CLOCK\_PERIOD;

-- test case 4: 15 + 15 + 1 = 31 (15)

A <= "1111"; B <= "1111"; cin <= '1';

wait for CLOCK\_PERIOD;

wait;

end process;

assertions: process

begin

wait for 2\*CLOCK\_PERIOD; -- reset time

wait for 4\*CLOCK\_PERIOD; -- 4 stages of pipeline

assert sum = "1111" and cout = '0'

report "Test case 1 failed (5 + 10 = 15)"

severity error;

wait for CLOCK\_PERIOD;

assert sum = "1011" and cout = '0'

report "Test case 2 failed (3 + 7 + 1 = 11)"

severity error;

wait for CLOCK\_PERIOD;

assert sum = "1011" and cout = '1'

report "Test case 3 failed (12 + 15 = 27 (11))"

severity error;

wait for CLOCK\_PERIOD;

assert sum = "1111" and cout = '1'

report "Test case 4 failed (15 + 15 + 1 = 31 (15))"

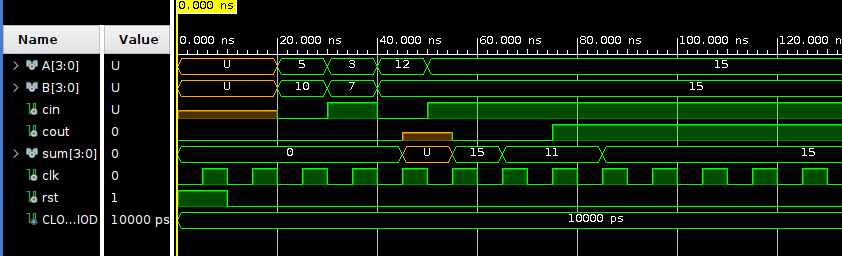
severity error;

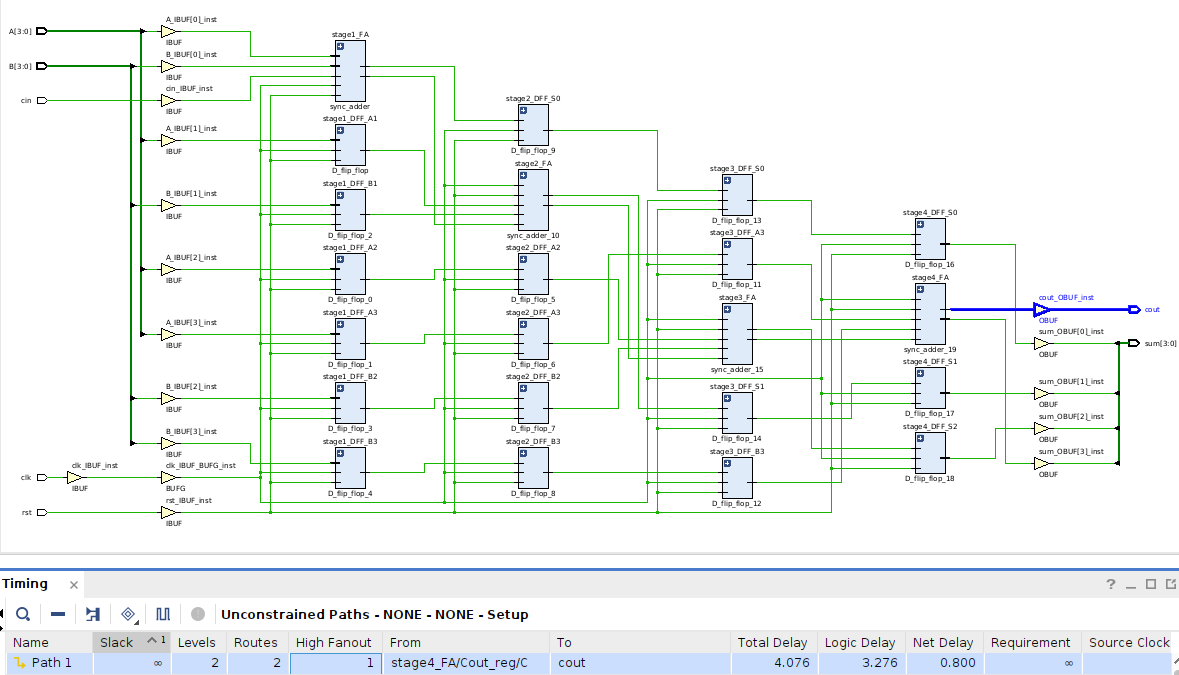
wait for CLOCK\_PERIOD;

wait;

end process;

end testbench;





*Critical Path (Stage 4 full adder -> cout): 4.076 ns*

## Systolic Multiplier

### Pipeline cell for multiplier

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity pipeline\_cell is

Port (

A : in std\_logic;

B : in std\_logic;

cin : in std\_logic;

sin : in std\_logic;

cout : out std\_logic;

sout : out std\_logic;

clk : in std\_logic;

rst : in std\_logic

);

end pipeline\_cell;

architecture structural of pipeline\_cell is

component sync\_adder is

Port (A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

rst : in std\_logic;

clk : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end component;

signal tmp : std\_logic;

begin

tmp <= A AND B;

fa: sync\_adder

port map (

A => tmp,

B => sin,

cin => cin,

rst => rst,

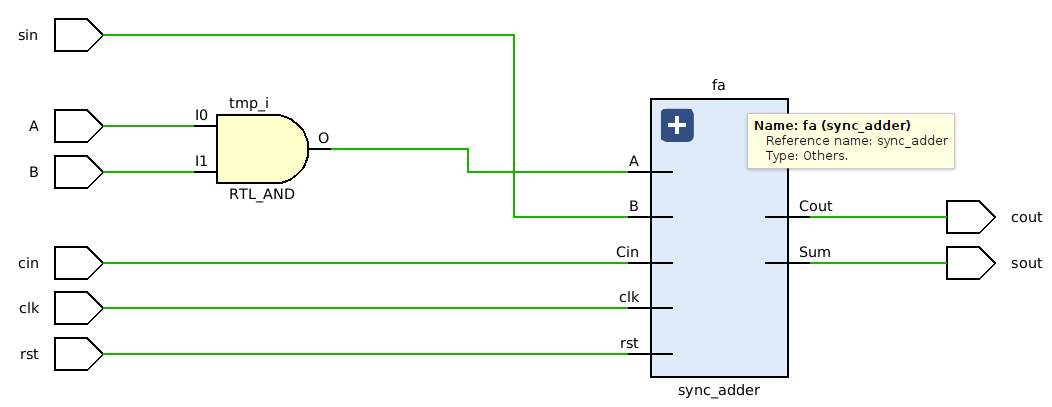
clk => clk,

sum => sout,

cout => cout

);

end structural;



### Systolic Multiplier Entity and Architecture Definition

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity systolic\_multiplier is

Port (

A : in STD\_LOGIC\_VECTOR(3 downto 0);

B : in STD\_LOGIC\_VECTOR(3 downto 0);

P : out STD\_LOGIC\_VECTOR(7 downto 0);

clk : in STD\_LOGIC;

rst : in STD\_LOGIC

);

end systolic\_multiplier;

architecture structural of systolic\_multiplier is

component pipeline\_cell is

Port (

A : in std\_logic;

B : in std\_logic;

cin : in std\_logic;

sin : in std\_logic;

cout : out std\_logic;

sout : out std\_logic;

clk : in std\_logic;

rst : in std\_logic

);

end component;

signal r0 : STD\_LOGIC\_VECTOR(8-1 downto 0); -- input register

signal r1, r2, r3 : STD\_LOGIC\_VECTOR(9-1 downto 0);

signal r4 : STD\_LOGIC\_VECTOR(10-1 downto 0);

signal r5, r6 : STD\_LOGIC\_VECTOR(9-1 downto 0);

signal r7 : STD\_LOGIC\_VECTOR(7-1 downto 0);

signal r8 : STD\_LOGIC\_VECTOR(8-1 downto 0);

signal r9 : STD\_LOGIC\_VECTOR(6-1 downto 0); -- output register

type matrix\_4x4 is array (0 to 3, 0 to 3) of std\_logic;

signal carrys : matrix\_4x4;

type matrix\_3x3 is array (0 to 3, 0 to 3) of std\_logic;

signal sum : matrix\_3x3;

signal p0,p1,p2,p3,p4,p5,p6,p7 : std\_logic;

begin

pc00: pipeline\_cell

port map(A => A(0), B => B(0), cin => '0', sin => '0', cout => carrys(0, 0), sout => p0, clk => clk, rst => rst);

pc01: pipeline\_cell

port map(A => r0(1), B => r0(4), cin => carrys(0, 0), sin => '0', cout => carrys(0, 1), sout => sum(0, 1), clk => clk, rst => rst);

pc02: pipeline\_cell

port map(A => r1(2), B => r1(4), cin => carrys(0, 1), sin => '0', cout => carrys(0, 2), sout => sum(0, 2), clk => clk, rst => rst);

pc03: pipeline\_cell

port map(A => r2(3), B => r2(4), cin => carrys(0, 2), sin => '0', cout => carrys(0, 3), sout => sum(0, 3), clk => clk, rst => rst);

pc10: pipeline\_cell

port map(A => r1(0), B => r1(5), cin => '0', sin => sum(0, 1), cout => carrys(1, 0), sout => p1, clk => clk, rst => rst);

pc11: pipeline\_cell

port map(A => r2(1), B => r2(5), cin => carrys(1, 0), sin => sum(0, 2), cout => carrys(1, 1), sout => sum(1, 1), clk => clk, rst => rst);

pc12: pipeline\_cell

port map(A => r3(2), B => r3(4), cin => carrys(1, 1), sin => sum(0, 3), cout => carrys(1, 2), sout => sum(1, 2), clk => clk, rst => rst);

pc13: pipeline\_cell

port map(A => r4(3), B => r4(4), cin => carrys(1, 2), sin => r4(9), cout => carrys(1, 3), sout => sum(1, 3), clk => clk, rst => rst);

pc20: pipeline\_cell

port map(A => r3(0), B => r3(5), cin => '0', sin => sum(1, 1), cout => carrys(2, 0), sout => p2, clk => clk, rst => rst);

pc21: pipeline\_cell

port map(A => r4(1), B => r4(5), cin => carrys(2, 0), sin => sum(1, 2), cout => carrys(2, 1), sout => sum(2, 1), clk => clk, rst => rst);

pc22: pipeline\_cell

port map(A => r5(2), B => r5(4), cin => carrys(2, 1), sin => sum(1, 3), cout => carrys(2, 2), sout => sum(2, 2), clk => clk, rst => rst);

pc23: pipeline\_cell

port map(A => r6(2), B => r6(3), cin => carrys(2, 2), sin => r6(8), cout => carrys(2, 3), sout => sum(2, 3), clk => clk, rst => rst);

pc30: pipeline\_cell

port map(A => r5(0), B => r5(5), cin => '0', sin => sum(2, 1), cout => carrys(3, 0), sout => p3, clk => clk, rst => rst);

pc31: pipeline\_cell

port map(A => r6(0), B => r6(4), cin => carrys(3, 0), sin => sum(2, 2), cout => carrys(3, 1), sout => p4, clk => clk, rst => rst);

pc32: pipeline\_cell

port map(A => r7(0), B => r7(2), cin => carrys(3, 1), sin => sum(2, 3), cout => carrys(3,2), sout => p5, clk => clk, rst => rst);

pc33: pipeline\_cell

port map(A => r8(0), B => r8(1), cin => carrys(3,2), sin => r8(7), cout => p7, sout => p6, clk => clk, rst => rst);

next\_clk : process(clk)

begin

if rising\_edge(clk) then

r9 <= p5 & r8(6 downto 2);

r8 <= carrys(2, 3) & p4 & r7(6 downto 1);

r7 <= p3 & r6(7 downto 4) & r6(2 downto 1);

r6 <= carrys(1, 3) & r5(8 downto 1);

r5 <= p2 & r4(8 downto 5) & r4(3 downto 0);

r4 <= carrys(0, 3) & r3(8 downto 0);

r3 <= p1 & r2(8 downto 5) & r2(3 downto 0);

r2 <= r1(8 downto 0);

r1 <= p0 & r0(7 downto 0);

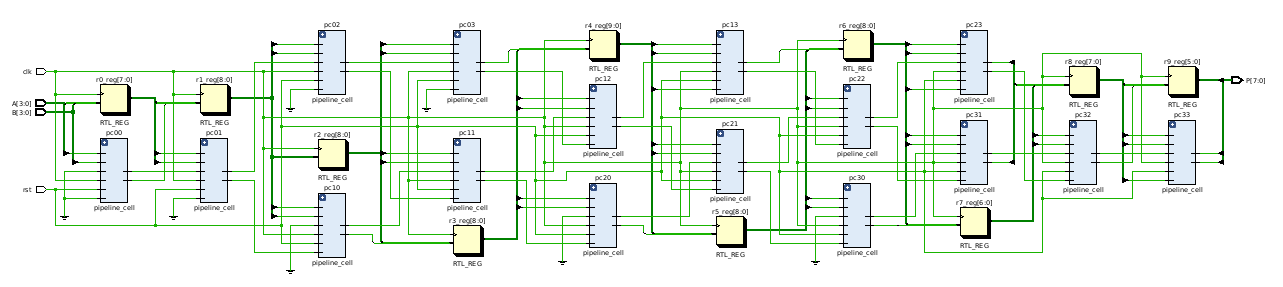
r0 <= B(3 downto 0) & A(3 downto 0);

end if;

P <= p7 & p6 & r9(5 downto 0);

end process;

end structural;



όπου τα κίτρινα κουτιά αναπαριστούν D Flip Flops ενώ τα μπλε pipeline cells

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use STD.TEXTIO.ALL;

entity systolic\_multiplier\_tb is

end systolic\_multiplier\_tb;

architecture test of systolic\_multiplier\_tb is

component systolic\_multiplier

port (

A: in std\_logic\_vector (3 downto 0);

B: in std\_logic\_vector (3 downto 0);

clk: in std\_logic;

rst: in std\_logic;

P: out std\_logic\_vector (7 downto 0)

);

end component;

signal A, B : std\_logic\_vector (3 downto 0):= (others=>'0');

signal clk, rst : std\_logic:='0';

signal P : std\_logic\_vector(7 downto 0);

constant CLOCK\_PERIOD : time := 10ns;

begin

UUT: systolic\_multiplier port map(

A => A,

B => B,

clk => clk,

rst => rst,

P => P (7 downto 0)

);

GEN\_CLK : process

begin

clk <= '0';

wait for CLOCK\_PERIOD/2;

clk <= '1';

wait for CLOCK\_PERIOD/2;

end process;

STIMULUS: process

begin

-- Test case 1: 5 \* 10 = 50

A <= "0101";

B <= "1010";

wait for CLOCK\_PERIOD;

-- Test case 2: 2 \* 4 = 8

A <= "0010";

B <= "0100";

wait for CLOCK\_PERIOD;

-- Test case 3: 12 \* 15 = 180

A <= "1100";

B <= "1111";

wait for CLOCK\_PERIOD;

-- Test case 5: 15 \* 15 = 225

A <= "1111";

B <= "1111";

wait for CLOCK\_PERIOD;

-- Test case 6: 1 \* 1 = 1

A <= "0001";

B <= "0001";

wait for CLOCK\_PERIOD;

-- Test case 7: 5 \* 12 = 60

A<="0101";

B<="1100";

wait for CLOCK\_PERIOD;

-- Test case 8: 6 \* 6 = 36

A<="0110";

B<="0110";

wait for CLOCK\_PERIOD;

-- Test case 9: 0 \* 15 = 0

A<="0000";

B<="1111";

wait for CLOCK\_PERIOD;

-- Test case 10: 4 \* 7 = 28

A<="0100";

B<="0111";

wait for CLOCK\_PERIOD;

A<="0000";

B<="0000";

wait;

end process;

assertions: process

begin

wait for 11 \* CLOCK\_PERIOD;

assert P = "00110010"

report "Test case 1 failed: 5 \* 10 expected 50 (00110010)"

severity error;

-- Test case 2: 2 \* 4 = 8

wait for CLOCK\_PERIOD;

assert P = "00001000"

report "Test case 2 failed: 2 \* 4 expected 8 (00001000)"

severity error;

-- Test case 3: 12 \* 15 = 180

wait for CLOCK\_PERIOD;

assert P = "10110100"

report "Test case 3 failed: 12 \* 15 expected 180 (10110100)"

severity error;

-- Test case 5: 15 \* 15 = 225

wait for CLOCK\_PERIOD;

assert P = "11100001"

report "Test case 5 failed: 15 \* 15 expected 225 (11100001)"

severity error;

-- Test case 6: 1 \* 1 = 1

wait for CLOCK\_PERIOD;

assert P = "00000001"

report "Test case 6 failed: 1 \* 1 expected 1 (00000001)"

severity error;

-- Test case 7: 5 \* 12 = 60

wait for CLOCK\_PERIOD;

assert P = "00111100"

report "Test case 7 failed: 5 \* 12 expected 60 (00111100)"

severity error;

-- Test case 8: 6 \* 6 = 36

wait for CLOCK\_PERIOD;

assert P = "00100100"

report "Test case 8 failed: 6 \* 6 expected 36 (00100100)"

severity error;

-- Test case 9: 0 \* 15 = 0

wait for CLOCK\_PERIOD;

assert P = "00000000"

report "Test case 9 failed: 0 \* 15 expected 0 (00000000)"

severity error;

-- Test case 10: 4 \* 7 = 28

wait for CLOCK\_PERIOD;

assert P = "00011101"

report "Test case 10 failed: 4 \* 7 expected 28 (00011100)"

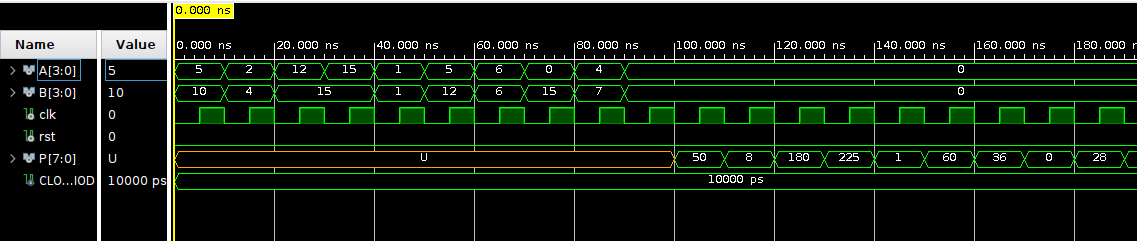
severity error;

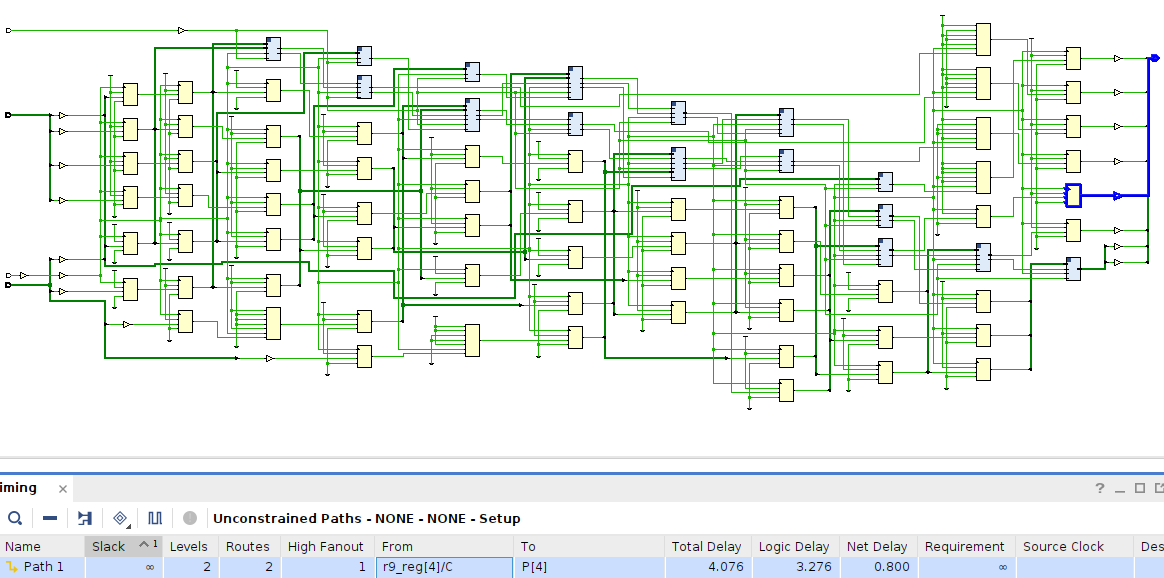
report "Finished Tests";

wait;

end process;

end test;





*Critical Path (Last Stage D Flip Flop -> P - out): 4.076 ns*