3η Εργαστηριακή Άσκηση

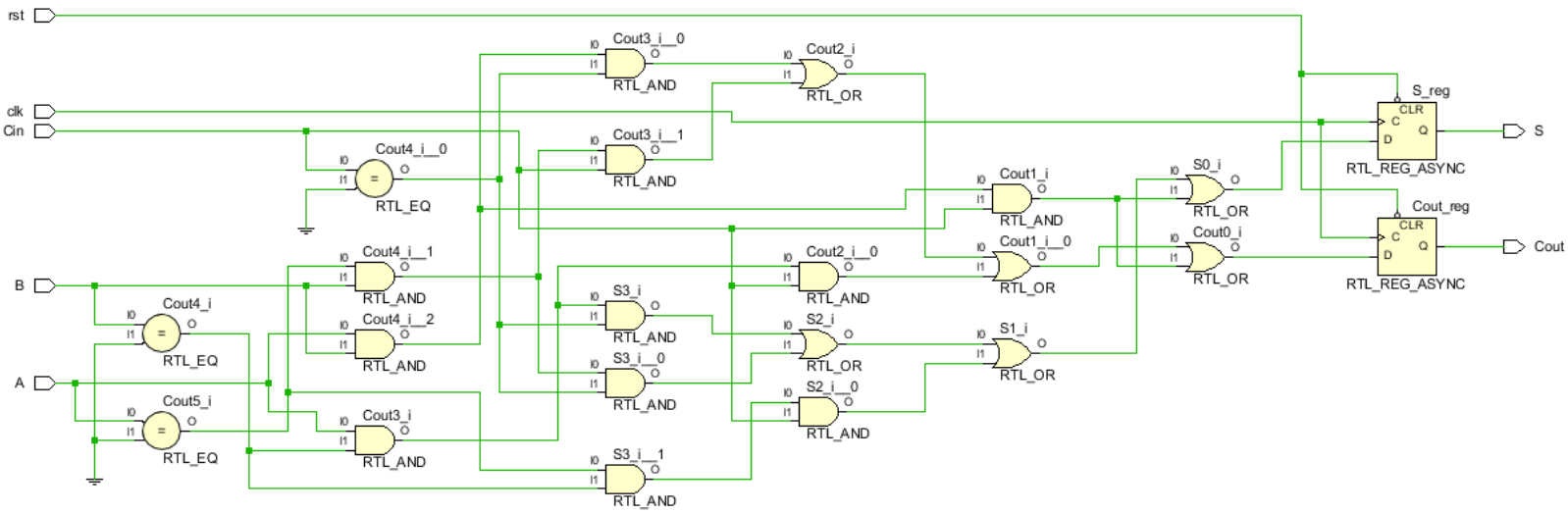
VLSI

Παντελαίος Δημήτριος Α.Μ.: 03118049

Μοίρας Αλέξανδρος Α.Μ.: 03118081

Ζήτημα 1:

Το RTL schematic του σύγχρονου πλήρη αθροιστή που υλοποιήσαμε:



Πρόκειται για ένα ακολουθιακό κύκλωμα, η έξοδος του οποίου αποτελείται από το αποτέλεσμα και το κρατούμενο της πρόσθεσης και ανανεώνεται στην θετική ακμή του ρολογιού. Το αποτέλεσμα της πρόθεσης είναι ‘1’, αν λαμβάνει την τιμή ‘1’ περιττό πλήθος εκ των σημάτων εισόδου A, B, $C_{in,}$ ενώ λαμβάνει την τιμή ‘0’ σε αντίθετη περίπτωση. Το κρατούμενο λαμβάνει την τιμή ‘1’, αν τουλάχιστον δύο από τα τρία σήματα εισόδου έχουν τιμή ‘1’. Ακόμη υπάρχει ένα σήμα reset, το οποίο όταν λαμβάνει την τιμή ‘0’ μηδενίζεται η έξοδος.

Ο VHDL κώδικας για την behavioral περιγραφή του σύγχρονου full adder:

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

entity full_adder_behavioral is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           S : out STD_LOGIC;
           Cout : out STD_LOGIC);
end full_adder_behavioral;

architecture Behavioral of full_adder_behavioral is

begin
    process(clk, rst)
    begin
        if rst='0' then
            S<='0';
            Cout<='0';
        elsif rising_edge(clk) then
            if ((A='1' and B='1' and Cin='0') or (A='0' and B='1' and Cin='1') or (A='1' and B='0' and Cin='1') or (A='1' and B='1' and Cin='1')) then
                Cout<='1';
            else
                Cout<='0';
            end if;

            if ((A='1' and B='0' and Cin='0') or (A='0' and B='1' and Cin='0') or (A='0' and B='0' and Cin='1') or (A='1' and B='1' and Cin='1')) then
                S<='1';
            else
                S<='0';
            end if;
        end if;
    end process;

end Behavioral;
```

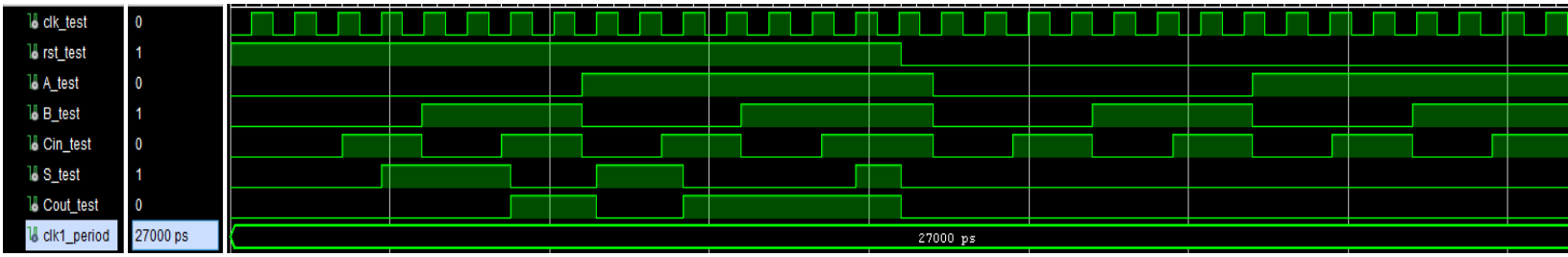
Ο VHDL κώδικας για το testbench της behavioral περιγραφής του σύγχρονου full adder:

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

entity full_adder_structural_testbench is
end full_adder_structural_testbench;

architecture test_full_adder of full_adder_structural_testbench is

constant clk1_period: time := 27ns;
signal clk_test, rst_test :std_logic := '0';
signal A_test, B_test, Cin_test :std_logic := '0';
signal S_test, Cout_test: std_logic;

component full_adder_behavioral is
    port(
        A, B, Cin, clk, rst: in std_logic;
        S, Cout: out std_logic
        );
end component;

begin
    uut: full_adder_behavioral
        port map(
            A=>A_test,
            B=>B_test,
            Cin => Cin_test,
            clk=>clk_test,
            rst=>rst_test,
            S=>S_test,
            Cout=>Cout_test
            );
    stimulus: process begin

        for i in 0 to 1 loop
            rst_test <= not rst_test;
            wait for 20ns;
```

```
            A_test<='0';
            B_test<='0';
            Cin_test<='0';
            wait for 50ns;
            A_test<='0';
            B_test<='0';
            Cin_test<='1';
            wait for 50ns;
            A_test<='0';
            B_test<='1';
            Cin_test<='0';
            wait for 50ns;
            A_test<='0';
            B_test<='1';
            Cin_test<='1';
            wait for 50ns;
            A_test<='1';
            B_test<='0';
            Cin_test<='0';
            wait for 50ns;
            A_test<='1';
            B_test<='0';
            Cin_test<='1';
            wait for 50ns;
            A_test<='1';
            B_test<='1';
            Cin_test<='0';
            wait for 50ns;
            A_test<='1';
            B_test<='1';
            Cin_test<='1';
            wait for 50ns;
        end loop;
    end process;

    clk1_generator: process begin
            clk_test <= '0';
            wait for clk1_period/2;
            clk_test <= '1';
            wait for clk1_period/2;
    end process;

end architecture;
```

Στο testbench δοκιμάζουμε όλες τις δυνατές τιμές για τα 3 σήματα εισόδου.

Η προσομοίωση που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός μας όπως περιεγράφη και παραπάνω:

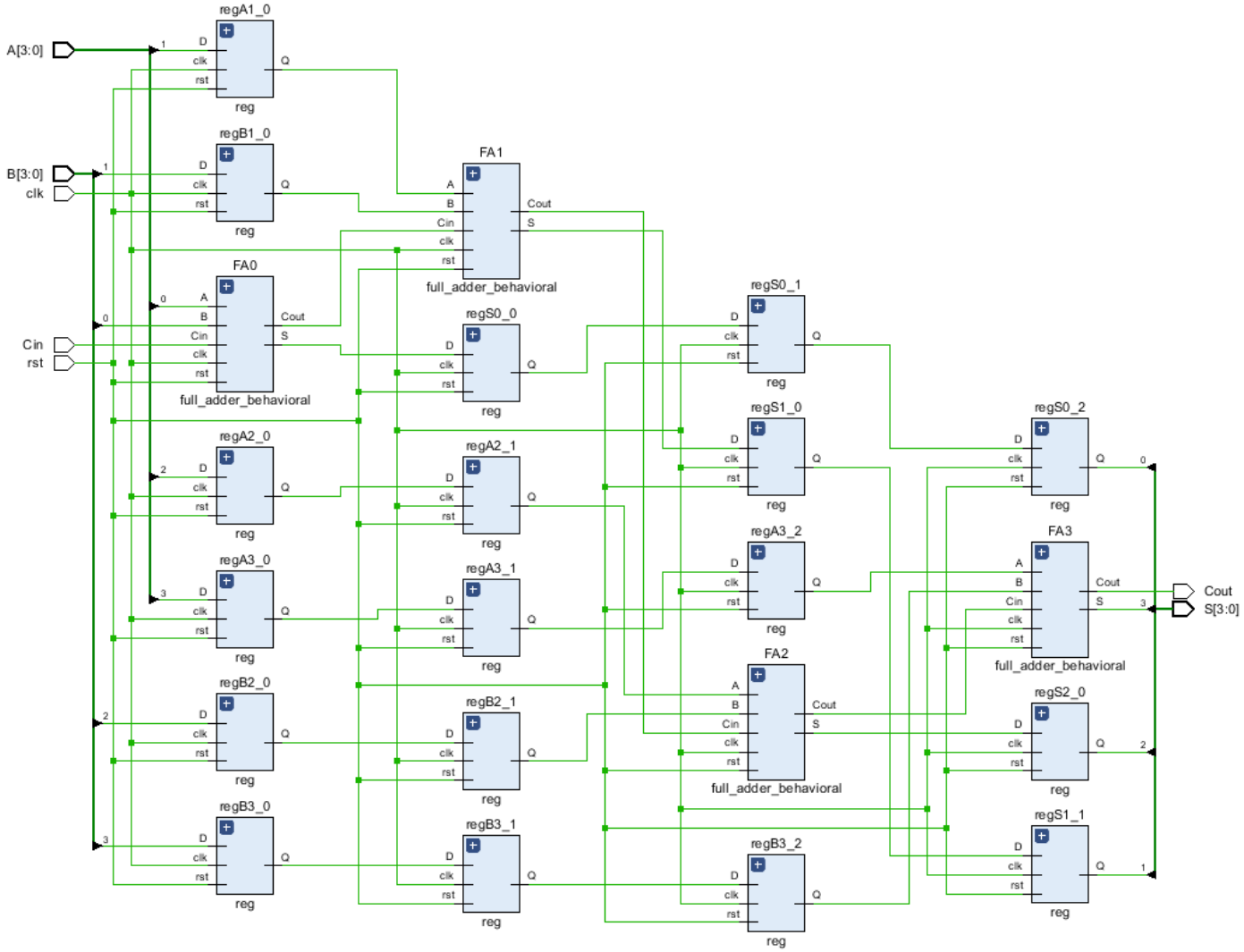


Στα ακολουθιακά κυκλώματα το κρίσιμο μονοπάτι προκύπτει από την είσοδο μέχρι κάποιον καταχωρητή ή από κάποιον καταχωρήτή μέχρι κάποιον άλλο ή από έναν καταχωρητή μέχρι την έξοδο.Το κρίσιμο μονοπάτι του κυκλώματος είναι από τους καταχωρητές Cout_reg και S_reg, που αποθηκεύεται το αποτέλεσμα της πρόσθεσης μέχρι την επόμενη θετική ακμή του ρολογιού προκειμένου να πάει στην έξοδο, μέχρι τις αντίστοιχες εξόδους Cout και S. Η χρονική του καθυστέρηση είναι 4.076ns.

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	2	2	1	Cout_reg/C	Cout	4.076	3.276	0.800	∞				0.000
Path 2	∞	2	2	1	S_reg/C	S	4.076	3.276	0.800	∞				0.000
Path 3	∞	2	3	2	rst	Cout_reg/CLR	2.693	1.106	1.587	∞	input port clock			0.000
Path 4	∞	2	3	2	rst	S_reg/CLR	2.693	1.106	1.587	∞	input port clock			0.000
Path 5	∞	2	3	2	B	S_reg/D	1.932	1.132	0.800	∞	input port clock			0.000
Path 6	∞	2	3	2	Cin	Cout_reg/D	1.906	1.106	0.800	∞	input port clock			0.000

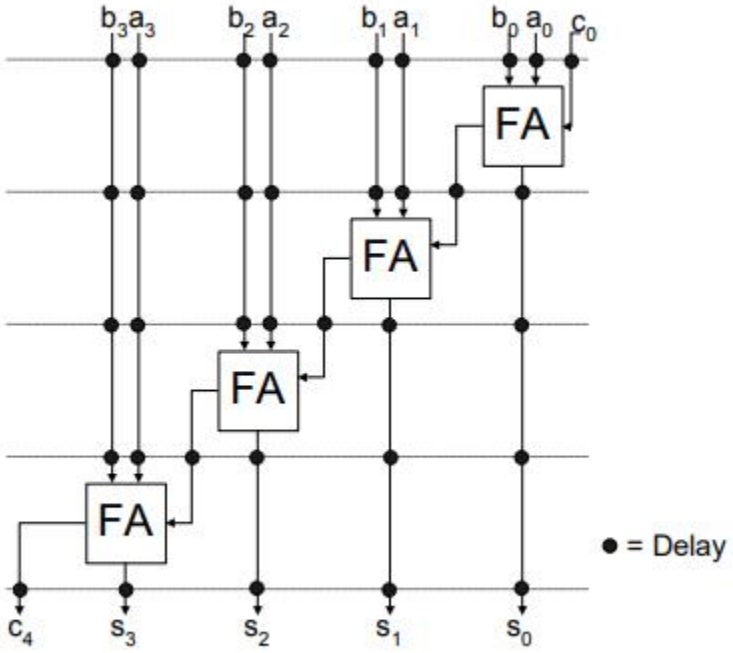
Ζήτημα 2:

Το RTL schematic του σύγχρονου αθροιστή διάδοσης κρατουμένου των 4 bits που υλοποιήσαμε:



Ο παραπάνω σύγχρονος αθροιστής διάδοσης κρατουμένου των 4 bits υλοποιήθηκε με χρήση της τεχνικής pipeline. Διαθέτουμε 4 σύγχρονους full adders του ερωτήματος 1. Ο κάθε full adder δίνει το αποτέλεσμα του στην έξοδο και το κρατούμενο του στον επόμενο full adder. Για να ολοκληρωθεί μια πρόθεση δύο αριθμών 4 bit χρειάζονται 4 κύκλοι, κατά τη διάρκεια των οποίων ο κάθε full adder χρησιμοποιείται διαδοχικά μόνο για έναν κύκλο. Συνεπώς όταν ολοκληρώνεται από τον full adder η πρόσθεση των αντίστοιχων ψηφίων δύο αριθμών, μπορούμε να του δώσουμε τα αντίστοιχα ψηφία της επόμενης πρόσθεσης δύο αριθμών που θέλουμε. Για να συμβεί αυτό όμως θα πρέπει να έχει παραχθεί το κρατούμενο από τον προηγούμενο full adder. Για αυτό το λόγο προσθέτουμε κατάλληλο αριθμό καταχωρητών, προκειμένου τα bit A_i, B_i να εισάγονται στον full adder όταν έχει παραχθεί το κρατούμενο C_{i-1}. Ακόμη προσθέτουμε καταχωρητές στην έξοδο των full adder, προκειμένου το αποτέλεσμα τους να φτάνει ταυτόχρονα

στην έξοδο. Οι καθυστερήσεις που αποτελούν έναν καταχωρητή στο κύκλωμα μας συμβολίζονται με τελεία στο παρακάτω σχήμα:



Σχήμα 1.13 Ο αθροιστής διάδοσης κρατουμένου σε λειτουργία συνεχούς διοχέτευσης

Ο VHDL κώδικας για την structural περιγραφή του σύγχρονου αθροιστή διάδοσης κρατουμένου των 4 bits με χρήση της τεχνικής Pipeline:

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


entity four_bit_FA_pipeline is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Cin : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Cout : out STD_LOGIC);
end four_bit_FA_pipeline;

architecture Structural of four_bit_FA_pipeline is

component reg is
    Port ( D : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Q : out STD_LOGIC);
end component;

component full_adder_behavioral is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           S : out STD_LOGIC;
           Cout : out STD_LOGIC);
end component;

--signal regCout, regCin : std_logic_vector(2 downto 0);
signal C1, C2, C3 : std_logic;
signal regA1, regB1 : std_logic;
signal regA2, regB2 : std_logic_vector(1 downto 0);
signal regA3, regB3 : std_logic_vector(2 downto 0);
signal regS0 : std_logic_vector (2 downto 0);
signal regS1 : std_logic_vector (1 downto 0);
signal regS2 : std_logic;
```

```
begin

FA0: full_adder_behavioral port map (A=>A(0), B=>B(0), Cin=>Cin, clk=>clk, rst=>rst, S=>regS0(0), Cout=>C1);
regS0_0: reg port map (D=>regS0(0), clk=>clk, rst=>rst, Q=>regS0(1));
regS0_1: reg port map (D=>regS0(1), clk=>clk, rst=>rst, Q=>regS0(2));
regS0_2: reg port map (D=>regS0(2), clk=>clk, rst=>rst, Q=>S(0));

regA1_0: reg port map (D=>A(1), clk=>clk, rst=>rst, Q=>regA1);
regB1_0: reg port map (D=>B(1), clk=>clk, rst=>rst, Q=>regB1);
FA1: full_adder_behavioral port map (A=>regA1, B=>regB1, Cin=>C1, clk=>clk, rst=>rst, S=>regS1(0), Cout=>C2);
regS1_0: reg port map (D=>regS1(0), clk=>clk, rst=>rst, Q=>regS1(1));
regS1_1: reg port map (D=>regS1(1), clk=>clk, rst=>rst, Q=>S(1));

regA2_0: reg port map (D=>A(2), clk=>clk, rst=>rst, Q=>regA2(0));
regB2_0: reg port map (D=>B(2), clk=>clk, rst=>rst, Q=>regB2(0));
regA2_1: reg port map (D=>regA2(0), clk=>clk, rst=>rst, Q=>regA2(1));
regB2_1: reg port map (D=>regB2(0), clk=>clk, rst=>rst, Q=>regB2(1));
FA2: full_adder_behavioral port map (A=>regA2(1), B=>regB2(1), Cin=>C2, clk=>clk, rst=>rst, S=>regS2, Cout=>C3);
regS2_0: reg port map (D=>regS2, clk=>clk, rst=>rst, Q=>S(2));

regA3_0: reg port map (D=>A(3), clk=>clk, rst=>rst, Q=>regA3(0));
regB3_0: reg port map (D=>B(3), clk=>clk, rst=>rst, Q=>regB3(0));
regA3_1: reg port map (D=>regA3(0), clk=>clk, rst=>rst, Q=>regA3(1));
regB3_1: reg port map (D=>regB3(0), clk=>clk, rst=>rst, Q=>regB3(1));
regA3_2: reg port map (D=>regA3(1), clk=>clk, rst=>rst, Q=>regA3(2));
regB3_2: reg port map (D=>regB3(1), clk=>clk, rst=>rst, Q=>regB3(2));
FA3: full_adder_behavioral port map (A=>regA3(2), B=>regB3(2), Cin=>C3, clk=>clk, rst=>rst, S=>S(3), Cout=>Cout);

end Structural;
```

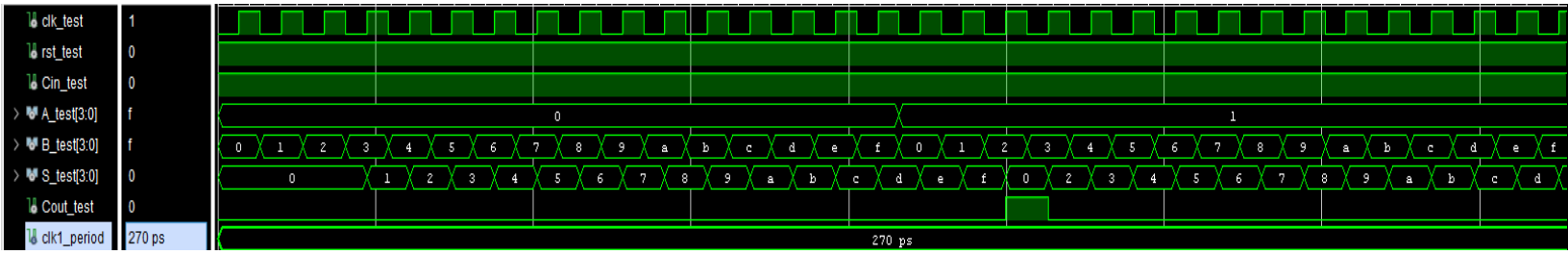
Ο VHDL κώδικας για το testbench της structural περιγραφής του σύγχρονου αθροιστή διάδοσης κρατουμένου των 4 bits με χρήση της τεχνικής Pipeline:

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

entity four_bit_FA_pipeline_testbench is
end four_bit_FA_pipeline_testbench;

architecture test_four_bit_full_adder of four_bit_FA_pipeline_testbench is

    constant clk1_period: time := 270ps;
    signal clk_test, rst_test :std_logic := '0';
    signal Cin_test :std_logic := '0';
    signal A_test, B_test :std_logic_vector(3 downto 0):= "1111";
    signal S_test :std_logic_vector(3 downto 0);
    signal Cout_test: std_logic;
```

```
    component four_bit_FA_pipeline is
        Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
               B : in STD_LOGIC_VECTOR (3 downto 0);
               Cin : in STD_LOGIC;
               clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               S : out STD_LOGIC_VECTOR (3 downto 0);
               Cout : out STD_LOGIC);
    end component;
begin
     uut: four_bit_FA_pipeline
        port map(
            A=>A_test,
            B=>B_test,
            Cin => Cin_test,
            clk=>clk_test,
            rst=>rst_test,
            S=>S_test,
            Cout=>Cout_test
            );
    stimulus: process begin
        for i in 0 to 1 loop
            rst_test <= not rst_test;
            for l in 0 to 1 loop
                Cin_test <= not Cin_test;
                for j in 0 to 15 loop
                    A_test<= A_test + 1;
                    for k in 0 to 15 loop
                        B_test <= B_test+1;
                        wait for clk1_period;
                    end loop;
                end loop;
            end loop;
        end loop;
        wait;
    end process;

    clk1_generator: process begin
            clk_test <= '0';
            wait for clk1_period/2;
            clk_test <= '1';
            wait for clk1_period/2;
    end process;
end test_four_bit_full_adder;
```

Στο testbench δοκιμάζουμε όλες τις δυνατές τιμές για τα 3 σήματα εισόδου.

Ένα μέρος της προσομοίωσης που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός μας όπως περιεγράφη και παραπάνω:

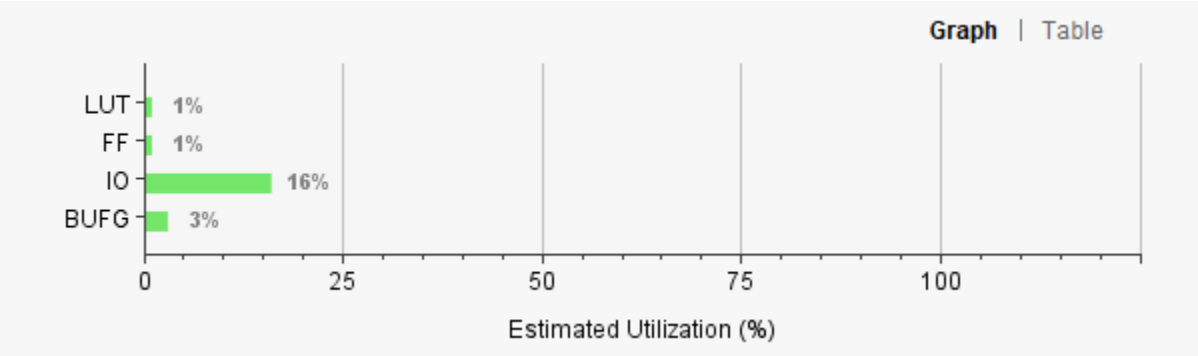


Η αρχική καθυστέρηση $T_{latency}$ είναι 4 κύκλοι ρολογιού μέχρι να παραχθεί το αποτέλεσμα της πρώτης πρόσθεσης που χρειάζεται 4 κύκλους. Μετά από αυτή την αρχική καθυστέρηση παράγεται ορθό αποτέλεσμα της αντίστοιχης πρόσθεσης σε κάθε κύκλο ρολογιού, καθώς χρησιμοποιήθηκε τεχνική pipeline και όταν ολοκληρώνεται μια πρόσθεση (από τον 4ο full adder), παράγεται ταυτόχρονα και το 3ο κρατούμενο της επόμενης πρόσθεσης από τον 3ο full adder και συνεπώς σε 1 κύκλο ολοκληρώνεται και αυτή η πρόσθεση.

Το κρίσιμο μονοπάτι του κυκλώματος είναι για τους 3 πρώτους full adder από το τελευταίο στάδιο καταχωρήτων μέχρι το αποτέλεσμα να βγει στην έξοδο και για τον 4ο full adder από την στιγμή που παράγεται το αποτέλεσμα μέχρι να βγει στην έξοδο. Η χρονική του καθυστέρηση είναι 4.076ns.

Path 1	∞	2	2	1	FA3/Cout_reg/C	Cout	4.076	3.276	0.800	∞				0.00
Path 2	∞	2	2	1	regS0_2/Q_reg/C	S[0]	4.076	3.276	0.800	∞				0.00
Path 3	∞	2	2	1	regS1_1/Q_reg/C	S[1]	4.076	3.276	0.800	∞				0.00
Path 4	∞	2	2	1	regS2_0/Q_reg/C	S[2]	4.076	3.276	0.800	∞				0.00
Path 5	∞	2	2	1	FA3/S_reg/C	S[3]	4.076	3.276	0.800	∞				0.00

Η κατανάλωση πόρων είναι η εξής:



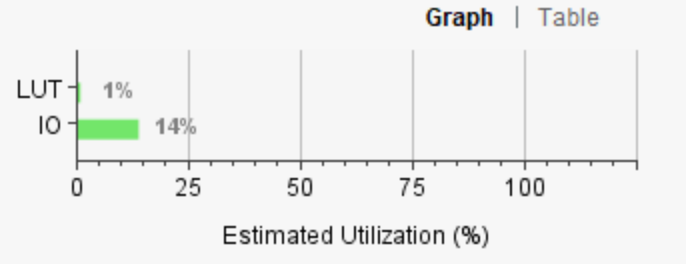
Resource	Estimation	Available	Utilization...
LUT	5	17600	0.03
FF	26	35200	0.07
IO	16	100	16.00
BUFG	1	32	3.13

Θεωρώντας τον Παράλληλο Αθροιστή της εργαστηριακής άσκησης 2 ως ένα σύγχρονο κύκλωμα του οποίου η χρονική καθυστέρηση του critical path αντιστοιχεί στην περίοδο ενός μεγάλου κύκλου ρολογιού, η περίοδος αυτού του ρολογιού θα ήταν τουλάχιστον 5.970ns.

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Sc
Path 1	∞	4	5	3	P4B[0]	P4S[2]	5.970	3.904	2.066	∞	in
Path 2	∞	4	5	3	P4B[0]	P4S[3]	5.970	3.904	2.066	∞	in
Path 3	∞	4	5	3	P4B[0]	PCout	5.964	3.898	2.066	∞	in
Path 4	∞	3	4	3	P4B[0]	P4S[0]	5.351	3.752	1.599	∞	in
Path 5	∞	3	4	2	P4B[1]	P4S[1]	5.351	3.752	1.599	∞	in

Για τον παράλληλο αθροιστή με pipeline που υλοποιήσαμε τώρα το critical path έχει καθυστέρηση 4.076ns επομένως η περιόδός του θεωρητικά μπορεί να είναι μικρότερη του παράλληλου αθροιστή της άσκησης 2. Αυτό συμβαίνει διότι ο παράλληλος αθροιστής της άσκησης 2 δε χρησιμοποιεί pipeline οπότε άπαξ και του δοθεί είσοδος χρειάζεται να αναμένει όλες τις βαθμίδες να ολοκληρώσουν τον υπολογισμό τους και να δώσουν στοιχεία στις επόμενες ώστε να παράξει ορθό αποτέλεσμα κάτι που είναι χρονοβόρο. Εν αντιθέσει στον pipelined παράλληλο αθροιστή, όπως εξηγήθηκε παραπάνω, το κρίσιμο μονοπάτι προκύπτει από την είσοδο μέχρι κάποιον καταχωρητή ή από κάποιον καταχωρητή μέχρι κάποιον άλλο ή από έναν καταχωρητή μέχρι την έξοδο, αφού πρόκειται για ακολουθιακό κύκλωμα, οπότε η καθυστέρηση υπολογίζεται για διάδοση δεδομένων από ένα βήμα στο επόμενο από κύκλο σε κύκλο ρολογιού και αυτή η καθυστέρηση αποτελεί κάτω όριο για την περίοδο του ρολογιού. Είναι λογικό λοιπόν να είναι μικρότερη από αυτήν του απλού παράλληλου αθροιστή η οποία αφορά ολόκληρο τον υπολογισμό του αθροίσματος. Οπότε θεωρώντας τον Παράλληλο Αθροιστή της εργαστηριακής άσκησης 2 ως ένα σύγχρονο κύκλωμα του οποίου η χρονική καθυστέρηση του critical path αντιστοιχεί στην περίοδο ενός μεγάλου κύκλου ρολογιού θα δίνει ορθό αποτέλεσμα με αλλαγή της εισόδου σε κάθε κύκλο ρολογιού σε περισσότερο χρόνο από ότι ο pipelined παράλληλος αθροιστής.

Όσον αφορά την κατανάλωση πόρων για τον παράλληλο αθροιστή της εργαστηριακής άσκησης 2 βλέπουμε τα εξής:

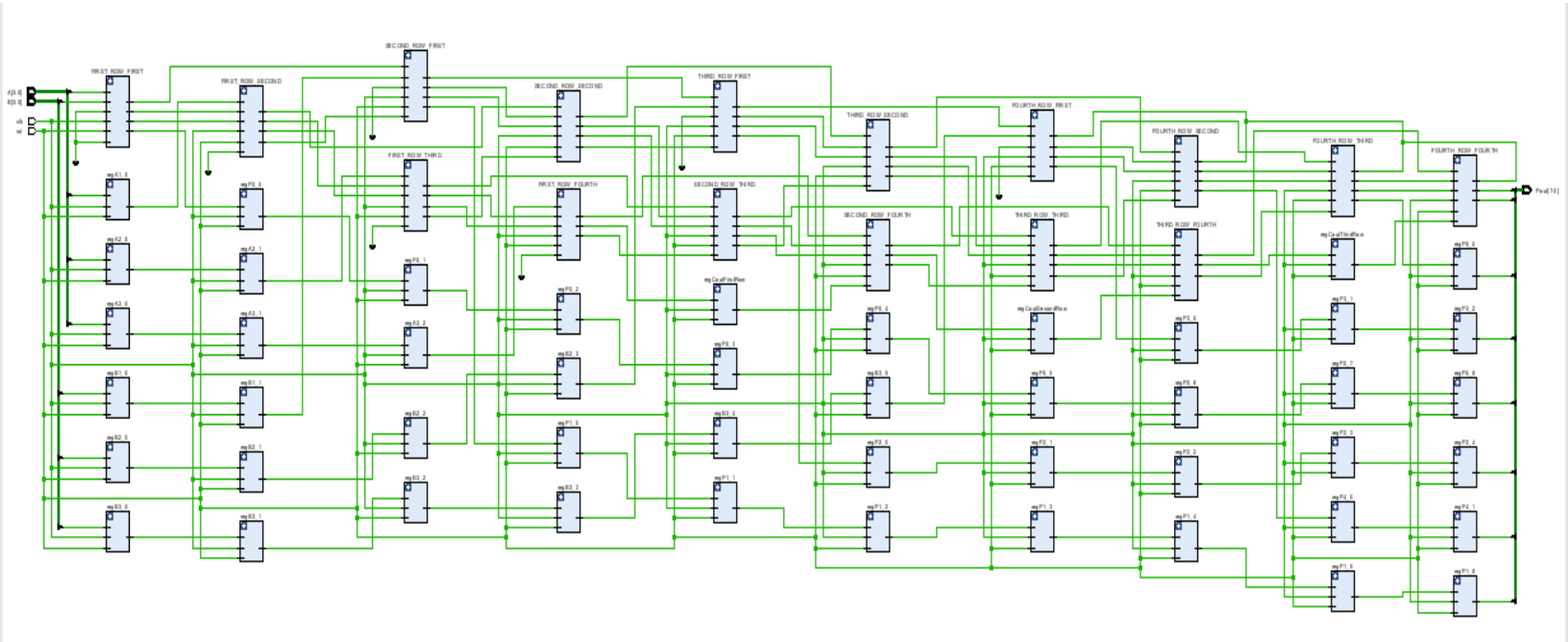


Resource	Estimation	Available	Utilization...
LUT	4	17600	0.02
IO	14	100	14.00

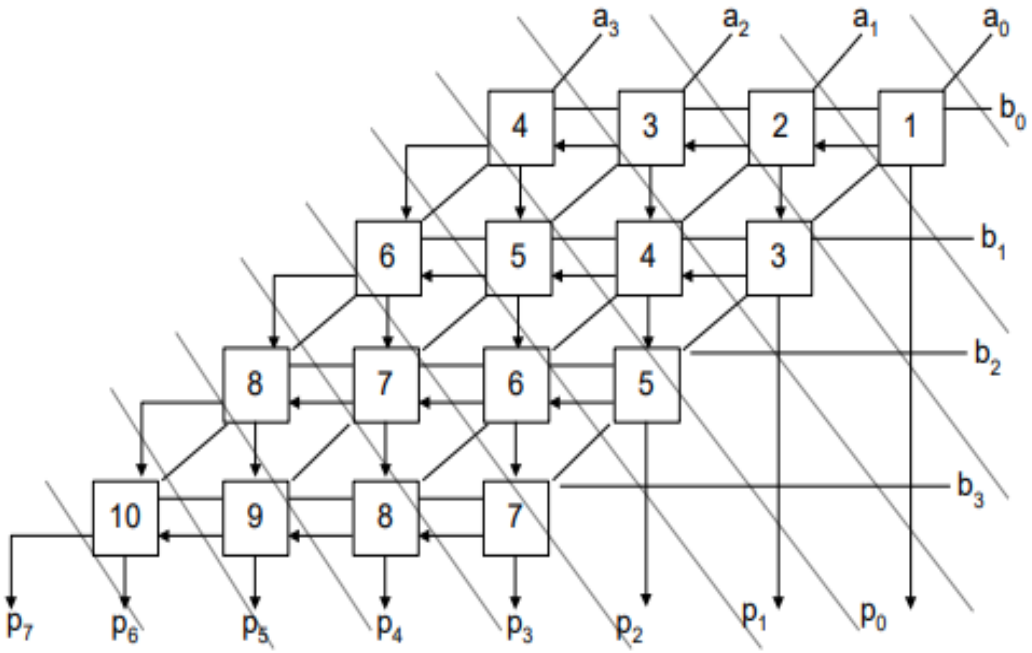
Παρατηρούμε ότι ο pipelined παράλληλος αθροιστής καταναλώνει σημαντικά περισσότερους πόρους, κάτι που είναι αναμενόμενο, καθώς χρειάζεται flip flops για τους registers που έχουμε εισάγει οι οποίοι αποτελούν επιπλέον hardware καθώς επίσης και global buffer για το ρολόι. Αλλαγές συνολικά στη δομή του κυκλώματος όπως η υλοποίηση των απλών πλήρων αθροιστών ως ακολουθιακά σύγχρονα κυκλώματα συμβάλουν και αυτές στην αύξηση της κατανάλωσης πόρων τόσο στα flip flops όσο και στα LUTs και IOs.

Ζήτημα 3:

Το RTL schematic του συστολικού πολλαπλασιαστή διάδοσης κρατουμένων των 4 bits που υλοποιήσαμε:



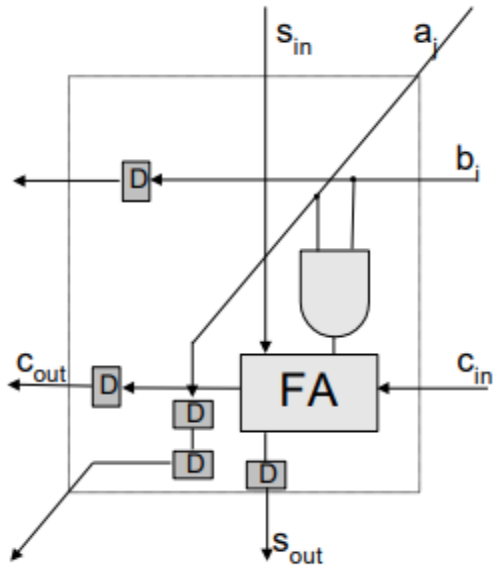
Χρησιμοποιήθηκαν επιπλέον καταχωρητές έτσι ώστε οι είσοδοι για την κάθε δομική μονάδα να φτάνουν ταυτόχρονα στους κατάλληλους κύκλους ρολογιού όπως φαίνεται στο παρακάτω σχήμα, ώστε να υπολογίζεται ορθά το αποτέλεσμα. Ακόμη χρησιμοποιήθηκαν καταχωρητές, ώστε όλα τα bit του τελικού αποτελέσματος να φτάνουν ταυτόχρονα στην έξοδο.



Σχήμα 1.33 Μετατροπή του ripple carry πολλαπλασιαστή σε συνεχούς διοχέτευσης και συστολικό

Οι αριθμοί πάνω στις δομικές μονάδες αντιπροσωπεύουν τον κύκλο στον οποίο πρέπει να υπολογίσει η καθεμία το αντίστοιχο αποτέλεσμα.

Η δομική μονάδα που χρησιμοποιήθηκε στο παραπάνω σχήμα είναι η παρακάτω αφαιρώντας του δύο καταχωρητές στην έξοδο του full adder, καθώς έγινε χρήση σύγχρονων full adder και δεν χρειάζεται η αποθήκευση του αποτελέσματος μέχρι την επόμενη θετική ακμή του ρολογιού:



Κάθε γραμμή του σχήματος αποτελεί τον πολλαπλασιασμό ενός bit του B με τον 4-bit A. Κατα τη διάρκεια του υπολογισμού γίνεται προώθηση του κρατουμένου και του bit του B, ενώ παράλληλα προωθούνται το αποτέλεσμα του πολλαπλασιασμού, το τελικό κρατούμενο και ο αριθμός A στην επόμενη γραμμή για τον επόμενο πολλαπλασιασμό.

Ο VHDL κώδικας για την structural περιγραφή του συστολικού πολλαπλασιαστή διάδοσης κρατουμένων των 4 bits:

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

entity multiplier_four_bit_pipeline is
    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;
           Pout : out STD_LOGIC_VECTOR (7 downto 0));
end multiplier_four_bit_pipeline;

architecture mult of multiplier_four_bit_pipeline is

    component full_adder_for_multiplier is
        Port ( a_j : in STD_LOGIC;
               b_i : in STD_LOGIC;
               cin : in STD_LOGIC;
               sin : in STD_LOGIC;
               clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               s_out : out STD_LOGIC;
               c_out : out STD_LOGIC;
               D_horizontal : out STD_LOGIC;
               D_diagonal : out STD_LOGIC);
    end component;

    component reg is
        Port ( D : in STD_LOGIC;
               clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               Q : out STD_LOGIC);
    end component;

signal result_first_row, carry_first_row, horizontal_first_row, diagonal_first_row : std_logic_vector(3 downto 0);
signal carry_buf_first_row, carry_buf_second_row, carry_buf_third_row : std_logic;
signal result_second_row, carry_second_row, horizontal_second_row, diagonal_second_row : std_logic_vector(3 downto 0);
signal result_third_row, carry_third_row, horizontal_third_row, diagonal_third_row : std_logic_vector(3 downto 0);
signal regP0 : std_logic_vector(8 downto 0);
signal carry_fourth_row, horizontal_fourth_row : std_logic_vector(3 downto 0);
signal dontcare : std_logic;
```

```
signal b1_buf: std_logic_vector(1 downto 0);
signal b2_buf: std_logic_vector(3 downto 0);
signal b3_buf: std_logic_vector(5 downto 0);

signal regP1 : std_logic_vector(6 downto 0);
signal regP2 : std_logic_vector(4 downto 0);
signal regP3 : std_logic_vector(2 downto 0);
signal regP4 : std_logic_vector(1 downto 0);
signal regP5 : std_logic;

signal a1_buf: std_logic;
signal a2_buf: std_logic_vector(1 downto 0);
signal a3_buf: std_logic_vector(2 downto 0);


begin

FIRST_ROW_FIRST: full_adder_for_multiplier port map(a_j=>A(0), b_i=>B(0), cin=>'0', sin=>'0', clk=>clk, rst=>rst, s_out=>regP0(0), c_out=>carry_first_row(0),
                                                    D_horizontal=>horizontal_first_row(0), D_diagonal=>diagonal_first_row(0));

regA1_0: reg port map (D=>A(1), clk=>clk, rst=>rst, Q=>a1_buf);

FIRST_ROW_SECOND: full_adder_for_multiplier port map(a_j=>a1_buf, b_i=>horizontal_first_row(0), cin=>carry_first_row(0), sin=>'0', clk=>clk, rst=>rst, s_out=>result_first_row(1), c_out=>carry_first_row(1),
                                                    D_horizontal=>horizontal_first_row(1), D_diagonal=>diagonal_first_row(1));

regA2_0: reg port map (D=>A(2), clk=>clk, rst=>rst, Q=>a2_buf(0));
regA2_1: reg port map (D=>a2_buf(0), clk=>clk, rst=>rst, Q=>a2_buf(1));

FIRST_ROW_THIRD: full_adder_for_multiplier port map(a_j=>a2_buf(1), b_i=>horizontal_first_row(1), cin=>carry_first_row(1), sin=>'0', clk=>clk, rst=>rst, s_out=>result_first_row(2), c_out=>carry_first_row(2),
                                                    D_horizontal=>horizontal_first_row(2), D_diagonal=>diagonal_first_row(2));

regA3_0: reg port map (D=>A(3), clk=>clk, rst=>rst, Q=>a3_buf(0));
regA3_1: reg port map (D=>a3_buf(0), clk=>clk, rst=>rst, Q=>a3_buf(1));
regA3_2: reg port map (D=>a3_buf(1), clk=>clk, rst=>rst, Q=>a3_buf(2));

FIRST_ROW_FOURTH: full_adder_for_multiplier port map(a_j=>a3_buf(2), b_i=>horizontal_first_row(2), cin=>carry_first_row(2), sin=>'0', clk=>clk, rst=>rst, s_out=>result_first_row(3), c_out=>carry_first_row(3),
                                                    D_horizontal=>horizontal_first_row(3), D_diagonal=>diagonal_first_row(3));

regCoutFirstRow: reg port map (D=>carry_first_row(3), clk=>clk, rst=>rst, Q=>carry_buf_first_row);
```

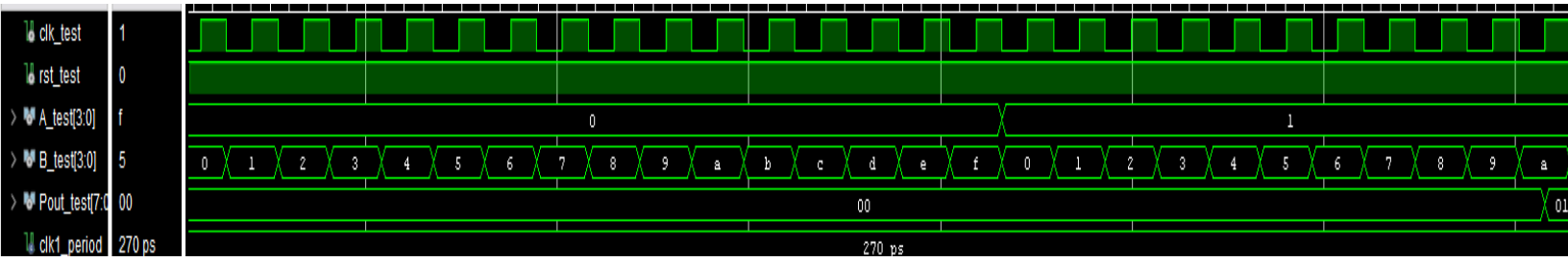
```
regP0_0: reg port map (D=>regP0(0), clk=>clk, rst=>rst, Q=>regP0(1));
regP0_1: reg port map (D=>regP0(1), clk=>clk, rst=>rst, Q=>regP0(2));
regP0_2: reg port map (D=>regP0(2), clk=>clk, rst=>rst, Q=>regP0(3));
regP0_3: reg port map (D=>regP0(3), clk=>clk, rst=>rst, Q=>regP0(4));
regP0_4: reg port map (D=>regP0(4), clk=>clk, rst=>rst, Q=>regP0(5));
regP0_5: reg port map (D=>regP0(5), clk=>clk, rst=>rst, Q=>regP0(6));
regP0_6: reg port map (D=>regP0(6), clk=>clk, rst=>rst, Q=>regP0(7));
regP0_7: reg port map (D=>regP0(7), clk=>clk, rst=>rst, Q=>regP0(8));
regP0_8: reg port map (D=>regP0(8), clk=>clk, rst=>rst, Q=>Pout(0));

regB1_0: reg port map (D=>B(1), clk=>clk, rst=>rst, Q=>b1_buf(0));
regB1_1: reg port map (D=>b1_buf(0), clk=>clk, rst=>rst, Q=>b1_buf(1));

SECOND_ROW_FIRST: full_adder_for_multiplier port map(a_j=>diagonal_first_row(0), b_i=>b1_buf(1), cin=>'0', sin=>result_first_row(1), clk=>clk, rst=>rst, s_out=>regP1(0), c_out=>carry_second_row(0),
                                                     D_horizontal=>horizontal_second_row(0), D_diagonal=>diagonal_second_row(0));
SECOND_ROW_SECOND: full_adder_for_multiplier port map(a_j=>diagonal_first_row(1), b_i=>horizontal_second_row(0), cin=>carry_second_row(0), sin=>result_first_row(2), clk=>clk, rst=>rst, s_out=>result_second_row(1),
                                                     c_out=>carry_second_row(1), D_horizontal=>horizontal_second_row(1), D_diagonal=>diagonal_second_row(1));
SECOND_ROW_THIRD: full_adder_for_multiplier port map(a_j=>diagonal_first_row(2), b_i=>horizontal_second_row(1), cin=>carry_second_row(1), sin=>result_first_row(3), clk=>clk, rst=>rst, s_out=>result_second_row(2),
                                                     c_out=>carry_second_row(2), D_horizontal=>horizontal_second_row(2), D_diagonal=>diagonal_second_row(2));
SECOND_ROW_FOURTH: full_adder_for_multiplier port map(a_j=>diagonal_first_row(3), b_i=>horizontal_second_row(2), cin=>carry_second_row(2), sin=>carry_buf_first_row, clk=>clk, rst=>rst, s_out=>result_second_row(3),
                                                     c_out=>carry_second_row(3), D_horizontal=>horizontal_second_row(3), D_diagonal=>diagonal_second_row(3));

regCoutSecondRow: reg port map (D=>carry_second_row(3), clk=>clk, rst=>rst, Q=>carry_buf_second_row);

regP1_0: reg port map (D=>regP1(0), clk=>clk, rst=>rst, Q=>regP1(1));
regP1_1: reg port map (D=>regP1(1), clk=>clk, rst=>rst, Q=>regP1(2));
regP1_2: reg port map (D=>regP1(2), clk=>clk, rst=>rst, Q=>regP1(3));
regP1_3: reg port map (D=>regP1(3), clk=>clk, rst=>rst, Q=>regP1(4));
regP1_4: reg port map (D=>regP1(4), clk=>clk, rst=>rst, Q=>regP1(5));
regP1_5: reg port map (D=>regP1(5), clk=>clk, rst=>rst, Q=>regP1(6));
regP1_6: reg port map (D=>regP1(6), clk=>clk, rst=>rst, Q=>Pout(1));

regB2_0: reg port map (D=>B(2), clk=>clk, rst=>rst, Q=>b2_buf(0));
regB2_1: reg port map (D=>b2_buf(0), clk=>clk, rst=>rst, Q=>b2_buf(1));
regB2_2: reg port map (D=>b2_buf(1), clk=>clk, rst=>rst, Q=>b2_buf(2));
regB2_3: reg port map (D=>b2_buf(2), clk=>clk, rst=>rst, Q=>b2_buf(3));

THIRD_ROW_FIRST: full_adder_for_multiplier port map(a_j=>diagonal_second_row(0), b_i=>b2_buf(3), cin=>'0', sin=>result_second_row(1), clk=>clk, rst=>rst, s_out=>regP2(0),
                                                    c_out=>carry_third_row(0), D_horizontal=>horizontal_third_row(0), D_diagonal=>diagonal_third_row(0));
THIRD_ROW_SECOND: full_adder_for_multiplier port map(a_j=>diagonal_second_row(1), b_i=>horizontal_third_row(0), cin=>carry_third_row(0), sin=>result_second_row(2), clk=>clk, rst=>rst, s_out=>result_third_row(1),
                                                    c_out=>carry_third_row(1), D_horizontal=>horizontal_third_row(1), D_diagonal=>diagonal_third_row(1));
```

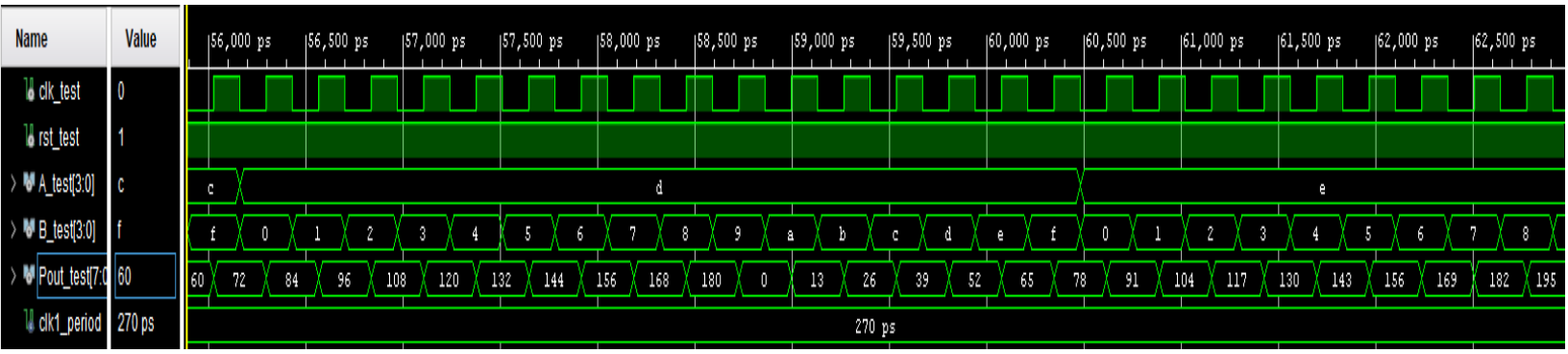
```
THIRD_ROW_THIRD: full_adder_for_multiplier port map(a_j=>diagonal_second_row(2), b_i=>horizontal_third_row(1), cin=>carry_third_row(1), sin=>result_second_row(3), clk=>clk, rst=>rst, s_out=>result_third_row(2),
                                                    c_out=>carry_third_row(2), D_horizontal=>horizontal_third_row(2), D_diagonal=>diagonal_third_row(2));
THIRD_ROW_FOURTH: full_adder_for_multiplier port map(a_j=>diagonal_second_row(3), b_i=>horizontal_third_row(2), cin=>carry_third_row(2), sin=>carry_buf_second_row, clk=>clk, rst=>rst, s_out=>result_third_row(3),
                                                    c_out=>carry_third_row(3), D_horizontal=>horizontal_third_row(3), D_diagonal=>diagonal_third_row(3));

regCoutThirdRow: reg port map (D=>carry_third_row(3), clk=>clk, rst=>rst, Q=>carry_buf_third_row);

regP2_0: reg port map (D=>regP2(0), clk=>clk, rst=>rst, Q=>regP2(1));
regP2_1: reg port map (D=>regP2(1), clk=>clk, rst=>rst, Q=>regP2(2));
regP2_2: reg port map (D=>regP2(2), clk=>clk, rst=>rst, Q=>regP2(3));
regP2_3: reg port map (D=>regP2(3), clk=>clk, rst=>rst, Q=>regP2(4));
regP2_4: reg port map (D=>regP2(4), clk=>clk, rst=>rst, Q=>Pout(2));

regB3_0: reg port map (D=>B(3), clk=>clk, rst=>rst, Q=>b3_buf(0));
regB3_1: reg port map (D=>b3_buf(0), clk=>clk, rst=>rst, Q=>b3_buf(1));
regB3_2: reg port map (D=>b3_buf(1), clk=>clk, rst=>rst, Q=>b3_buf(2));
regB3_3: reg port map (D=>b3_buf(2), clk=>clk, rst=>rst, Q=>b3_buf(3));
regB3_4: reg port map (D=>b3_buf(3), clk=>clk, rst=>rst, Q=>b3_buf(4));
regB3_5: reg port map (D=>b3_buf(4), clk=>clk, rst=>rst, Q=>b3_buf(5));

FOURTH_ROW_FIRST: full_adder_for_multiplier port map(a_j=>diagonal_third_row(0), b_i=>b3_buf(5), cin=>'0', sin=>result_third_row(1), clk=>clk, rst=>rst, s_out=>regP3(0), c_out=>carry_fourth_row(0),
                                                    D_horizontal=>horizontal_fourth_row(0), D_diagonal=>dontcare);

regP3_0: reg port map (D=>regP3(0), clk=>clk, rst=>rst, Q=>regP3(1));
regP3_1: reg port map (D=>regP3(1), clk=>clk, rst=>rst, Q=>regP3(2));
regP3_2: reg port map (D=>regP3(2), clk=>clk, rst=>rst, Q=>Pout(3));

FOURTH_ROW_SECOND: full_adder_for_multiplier port map(a_j=>diagonal_third_row(1), b_i=>horizontal_fourth_row(0), cin=>carry_fourth_row(0), sin=>result_third_row(2), clk=>clk, rst=>rst, s_out=>regP4(0),
                                                     c_out=>carry_fourth_row(1), D_horizontal=>horizontal_fourth_row(1), D_diagonal=>dontcare);

regP4_0: reg port map (D=>regP4(0), clk=>clk, rst=>rst, Q=>regP4(1));
regP4_1: reg port map (D=>regP4(1), clk=>clk, rst=>rst, Q=>Pout(4));

FOURTH_ROW_THIRD: full_adder_for_multiplier port map(a_j=>diagonal_third_row(2), b_i=>horizontal_fourth_row(1), cin=>carry_fourth_row(1), sin=>result_third_row(3), clk=>clk, rst=>rst, s_out=>regP5,
                                                    c_out=>carry_fourth_row(2), D_horizontal=>horizontal_fourth_row(2), D_diagonal=>dontcare);

regP5_0: reg port map (D=>regP5, clk=>clk, rst=>rst, Q=>Pout(5));

FOURTH_ROW_FOURTH: full_adder_for_multiplier port map(a_j=>diagonal_third_row(3), b_i=>horizontal_fourth_row(2), cin=>carry_fourth_row(2), sin=>carry_buf_third_row, clk=>clk, rst=>rst, s_out=>Pout(6),
                                                     c_out=>Pout(7), D_horizontal=>horizontal_fourth_row(3), D_diagonal=>dontcare);

end mult;
```

Ο VHDL κώδικας για το την testbench της structural περιγραφής του συστολικού πολλαπλασιαστή διάδοσης κρατουμένων των 4 bits:

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

entity multiplier_four_bit_pipeline_testbench is
end multiplier_four_bit_pipeline_testbench;

architecture testbench_multiplier of multiplier_four_bit_pipeline_testbench is
    constant clkl_period: time := 270ps;
    signal clk_test, rst_test : std_logic := '0';
    signal A_test, B_test : std_logic_vector(3 downto 0) := "1111";
    signal Pout_test : std_logic_vector(7 downto 0);

    component multiplier_four_bit_pipeline is
        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;
               Pout : out STD_LOGIC_VECTOR (7 downto 0));
    end component;
begin
    uut: multiplier_four_bit_pipeline
        port map(
            A=>A_test,
            B=>B_test,
            clk=>clk_test,
            rst=>rst_test,
            Pout=>Pout_test);

    stimulus: process begin
        for i in 0 to 1 loop
            rst_test <= not rst_test;
            for j in 0 to 15 loop
                A_test<=A_test + 1;
                for k in 0 to 15 loop
                    B_test<=B_test + 1;
                    wait for clkl_period;
                end loop;
            end loop;
            wait for 10*clkl_period;
        end loop;
    end process;

    clkl_generator: process begin
            clk_test <= '0';
            wait for clkl_period/2;
            clk_test <= '1';
            wait for clkl_period/2;
    end process;

end testbench_multiplier;
```

Ένα μέρος της προσομοίωσης που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός μας όπως περιεγράφη και παραπάνω:





Η αρχική καθυστέρηση $T_{latency}$ είναι 10 κύκλοι ρολογιού μέχρι να παραχθεί το αποτέλεσμα του πρώτου πολλαπλασιασμού που χρειάζεται 10 κύκλους. Μετά από αυτή την αρχική καθυστέρηση παράγεται ορθό αποτέλεσμα του αντίστοιχου πολλαπλασιασμού σε κάθε κύκλο ρολογιού, καθώς χρησιμοποιήθηκε τεχνική συστολικού pipeline και όταν παράγεται το αποτέλεσμα ενός πολλαπλασιασμού στον 10 κύκλο χρησιμοποιώντας μόνο την τελευταία δομική μονάδα, έχουν παραχθεί ήδη οι είσοδοι που θα δοθούν σε αυτή την δομική μονάδα για την ολοκλήρωση του επόμενου πολλαπλασιασμού.

Το κρίσιμο μονοπάτι του κυκλώματος είναι από τους τελικούς καταχωρητές ή την δομική μονάδα που χρησιμοποιείται στον τελευταίο κύκλο μέχρι την έξοδο. Η χρονική του καθυστέρηση είναι 4.076ns.

Path 1	∞	2	2	1	regP0_8/Q_reg/C	Pout[0]	4.076
Path 2	∞	2	2	1	regP1_6/Q_reg/C	Pout[1]	4.076
Path 3	∞	2	2	1	regP2_4/Q_reg/C	Pout[2]	4.076
Path 4	∞	2	2	1	regP3_2/Q_reg/C	Pout[3]	4.076
Path 5	∞	2	2	1	regP4_1/Q_reg/C	Pout[4]	4.076
Path 6	∞	2	2	1	regP5_0/Q_reg/C	Pout[5]	4.076
Path 7	∞	2	2	1	FOURTH_ROW_FOURTH/fll_adder/S_reg/C	Pout[6]	4.076
Path 8	∞	2	2	1	FOURTH_ROW_FOURTH/fll_adder/Cout_reg/C	Pout[7]	4.076