Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

3η Σειρά Ασκήσεων

Μάθημα: Ψηφιακά Συστήματα VLSI

Εξάμηνο: 8ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Θέμα 1: Full Synchronous Adder

Στο πρώτο μέρος της εργαστηριακής άσκησης υλοποιούμε έναν σύχρονο Πλήρη Αθροιστή (Full Adder - FA) με περιγραφή συμπεριφοράς (Behavioral).

Ένας σύγχρονος πλήρης αθροιστής είναι ένα ψηφιακό κύκλωμα που προσθέτει τρία δυαδικά ψηφία (A, B και Cin) και παράγει ένα άθροισμα (S) και ένα κρατούμενο (Cout). Ο όρος "σύγχρονος" αναφέρεται στο γεγονός ότι όλες οι λειτουργίες εντός του αθροιστή συγχρονίζονται με ένα κοινό σήμα ρολογιού. Αυτό σημαίνει ότι οι είσοδοι δειγματοληπτούνται και υποβάλλονται σε επεξεργασία μόνο σε συγκεκριμένες άκρες ρολογιού (στη δικιά μας περίπτωση στην θετική ακμή), διασφαλίζοντας τον σωστό χρονισμό και τη λειτουργία του κυκλώματος.

Το άθροισμα (S) έξοδος του σύγχρονου πλήρους αθροιστή υπολογίζεται με βάση τα δυαδικά ψηφία εισόδου A, B και το κρατούμενο εισόδου (Cin). Το άθροισμα είναι «1» εάν ο συνολικός αριθμός των «1» μεταξύ των εισόδων (A, B, Cin) είναι περιττός και «0» διαφορετικά.

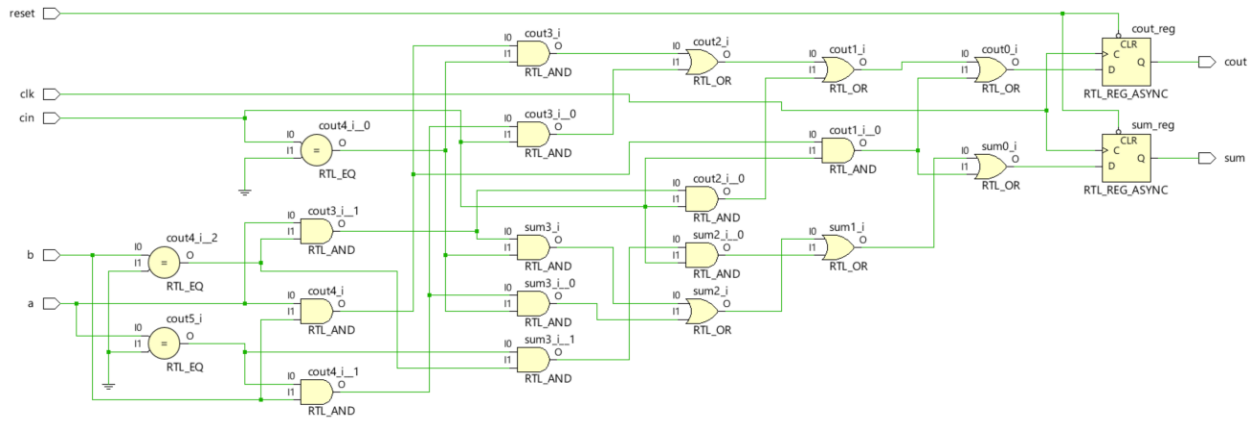
Η εκτέλεση (Cout) του σύγχρονου πλήρους αθροιστή προσδιορίζεται με βάση τα bit εισόδου A, B και Cin. Είναι «1» εάν τουλάχιστον δύο από τα σήματα εισόδου (A, B, Cin) είναι «1».

Ο κώδικας Behavioral αρχιτεκτονικής φαίνεται παρακάτω:

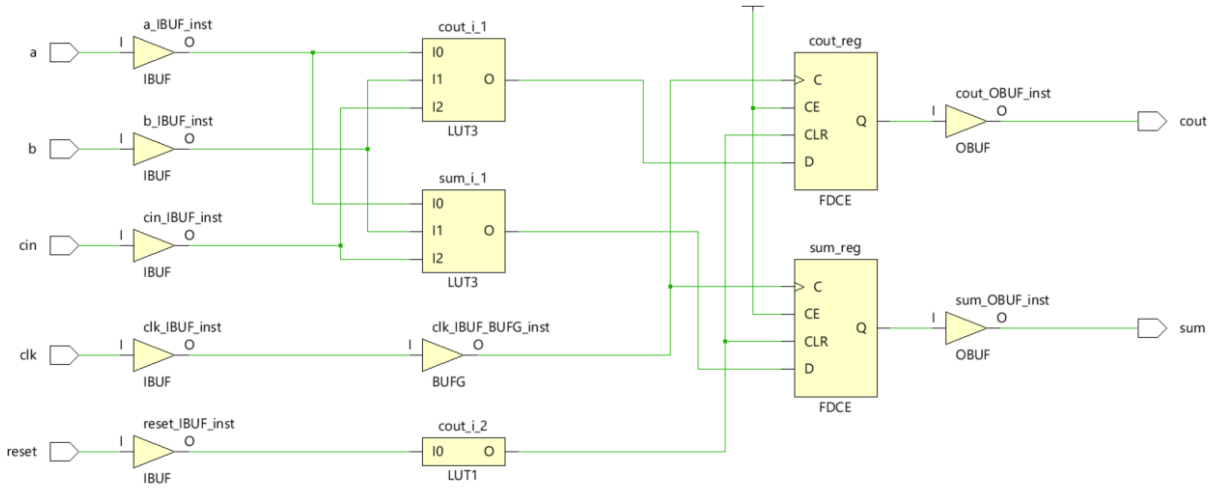
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity full_adder_bh is
    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 full_adder_bh;

architecture Behavioral of full_adder_bh is

begin
    process(clk, rst)
    begin
        if rst='0' then
            sum <= '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
                sum <= '1';
            else
                sum <= '0';
            end if;
        end if;
```

```
    end process;

end Behavioral;
```

Το RTL schematic του σύγχρονου πλήρη αθροιστή που υλοποιήσαμε:



Το Synthesis Schematic:



Το critical path ενός ακολουθιακού κυκλώματος, όπως ένας σύγχρονος πλήρης αθροιστής, είναι η μεγαλύτερη διαδρομή από την είσοδο ενός flip-flop στην έξοδο του ίδιου ή άλλου flip-flop. Αντιπροσωπεύει τη μέγιστη καθυστέρηση μέσω του κυκλώματος, η οποία καθορίζει τη μέγιστη συχνότητα λειτουργίας και τη συνολική απόδοση. Το κρίσιμο μονοπάτι του κυκλώματος είναι από τους καταχωρητές cout_reg και sum_reg που αποθηκεύεται το αποτέλεσνα της πρόσθεσης μέχρι την επόμενη

θετική ακμή του ρολογιού προκειμένου να πάει στην έξοδο, μέχρι τις αντίστοιχες εξόδους cout και sum. 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
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	rst	cout_reg/CLR	2.693	1.106	1.587	∞	input port clock
Path 4	∞	2	3	2	rst	sum_reg/CLR	2.693	1.106	1.587	∞	input port clock
Path 5	∞	2	3	2	b	sum_reg/D	1.932	1.132	0.800	∞	input port clock
Path 6	∞	2	3	2	cin	cout_reg/D	1.906	1.106	0.800	∞	input port clock

Εκτελώντας και την εντολή report_timing_summary -report_unconstrained στο terminal παίρνουμε μια πιο αναλυτική αναφορά για το μέγιστο Critical Path:

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack:                    inf
  Source:                 cout_reg/C
                            (rising edge-triggered cell FDCE)
  Destination:            cout
                            (output port)
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        4.076ns  (logic 3.276ns (80.380%)  route 0.800ns (19.620%))
  Logic Levels:           2  (FDCE=1 OBUF=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDCE                         0.000     0.000 r  cout_reg/C
                         FDCE (Prop fdce C Q)         0.456     0.456 r  cout reg/Q

                           net (fo=1, unplaced)          0.800     1.256    cout_OBUF
                           OBUF (Prop_obuf_I_O)          2.820     4.076 r  cout_OBUF_inst/O
                           net (fo=0)                    0.000     4.076    cout
                                                                         r  cout (OUT)
  ---------------------------------------------------------------------    -------------------
```

```
Min Delay Paths
--------------------------------------------------------------------------------------
Slack:                    inf
  Source:                 a
                            (input port)
  Destination:            sum_reg/D
```

```
Path Group:             (none)
Path Type:              Min at Fast Process Corner
Data Path Delay:        0.590ns  (logic 0.253ns (42.882%)  route 0.337ns (57.118%))
Logic Levels:           2  (IBUF=1 LUT3=1)

  Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
-------------------------------------------------------------------    -------------------
                                                  0.000     0.000 r  a (IN)
                     net (fo=0)                   0.000     0.000    a
                     IBUF (Prop_ibuf_I_O)         0.211     0.211 r  a_IBUF_inst/O
                     net (fo=2, unplaced)         0.337     0.548    a_IBUF
                     LUT3 (Prop_lut3_I0_O)        0.042     0.590 r  sum_i_1/O
                     net (fo=1, unplaced)         0.000     0.590    sum0
                     FDCE                                        r  sum_reg/D
-------------------------------------------------------------------    -------------------
```

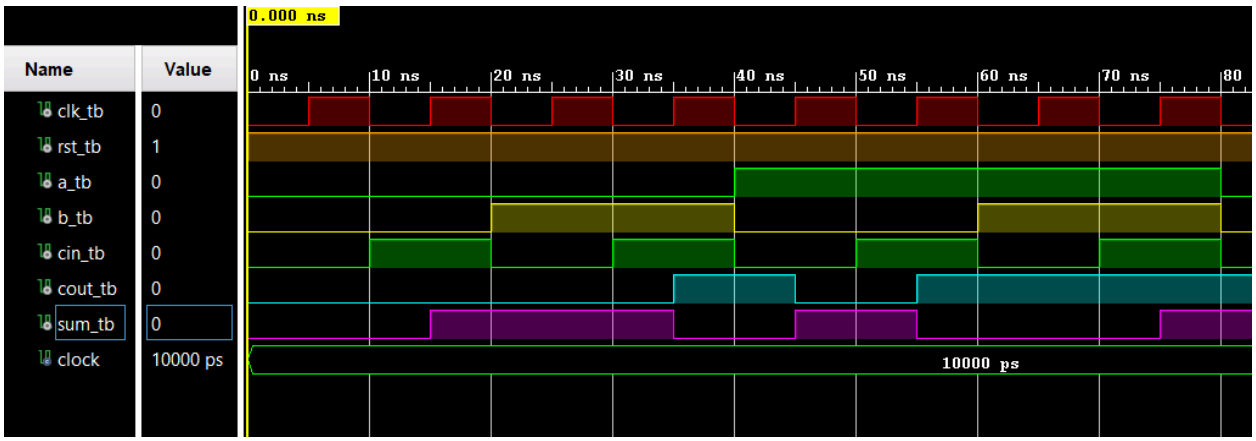
Ο κώδικας για το TestBench φαίνεται παρακάτω:

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

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

architecture bench of full_adder_bh_tb is
    component full_adder_bh is
        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;

    signal clk_tb, rst_tb, a_tb, b_tb, cin_tb : std_logic:='0';
    signal cout_tb, sum_tb : std_logic;

    constant clock : time := 10ns;

begin
```

```
dut: full_adder_bh
    port map (
        a => a_tb,
        b => b_tb,
        cin => cin_tb,
        clk => clk_tb,
        rst => rst_tb,
        sum => sum_tb,
        cout => cout_tb
    );

simulation : process
begin
    rst_tb <= '1';

    a_tb <= '0';
    b_tb <= '0';
    cin_tb <= '0';
    wait for clock;

    a_tb <= '0';
    b_tb <= '0';
    cin_tb <= '1';
    wait for clock;

    a_tb <= '0';
    b_tb <= '1';
    cin_tb <= '0';
    wait for clock;

    a_tb <= '0';
    b_tb <= '1';
    cin_tb <= '1';
    wait for clock;

    a_tb <= '1';
    b_tb <= '0';
    cin_tb <= '0';
    wait for clock;
```

```
        a_tb <= '1';
        b_tb <= '0';
        cin_tb <= '1';
        wait for clock;

        a_tb <= '1';
        b_tb <= '1';
        cin_tb <= '0';
        wait for clock;

        a_tb <= '1';
        b_tb <= '1';
        cin_tb <= '1';
        wait for clock;

    end process;

    generate_clock : process
    begin
        clk_tb <= '0';
        wait for clock/2;
        clk_tb <= '1';
        wait for clock/2;
    end process;

end bench;
```

To Simulation:



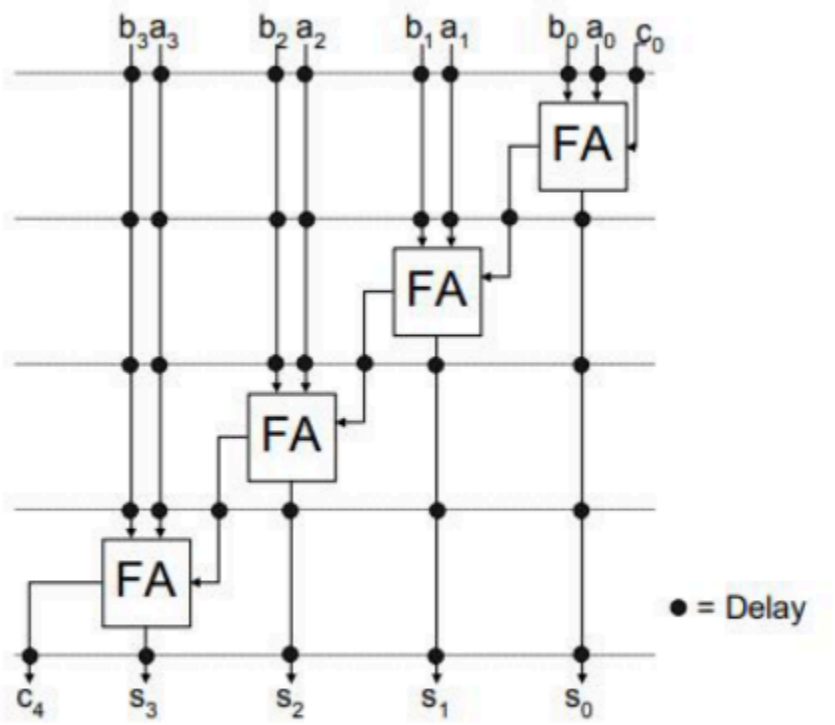
Η παραπάνω προσομοίωση επιβεβαιώνει την ορθή λειτουργία του κυκλώματος μας.

Θέμα 2: Synchronous FA with Pipeline

Στο δεύτερο ζητούμενο, πρέπει να υλοποιήσουμε έναν σύγχρονο Αθροιστή διάδοσης κρατουμένου των 4 bits με χρήση της τεχνικής Pipeline. Το κύκλωμα θα πρέπει να τροφοδοτείται με ένα διαφορετικό ζεύγος εισόδων σε κάθε κύκλο ρολογιού και να δίνει αντίστοιχα ορθό αποτέλεσμα σε κάθε κύκλο ρολογιού έπειτα από κάποια αρχική καθυστέρηση Tlatency . Η υλοποίηση να βασιστεί στη δομική μονάδα του Πλήρη Αθροιστή του Ερωτήματος 1 και για τους σκοπούς της άσκησης φτιάχνουμε ένα dff structure για τη διατήρηση τόσο των inputs, οσο και του αντίστοιχου output.

Για να εφαρμόσουμε έναν σύγχρονο αθροιστή διάδοσης κρατουμένου 4 bit χρησιμοποιώντας την τεχνική pipeline, θα χρησιμοποιήσουμε το δομικό στοιχείο Full Adder και θα εισαγάγουμε πρόσθετους καταχωρητές για τη διοχέτευση του υπολογισμού. Αυτή η προσέγγιση μας επιτρέπει να επεξεργαστούμε ένα διαφορετικό ζεύγος εισόδων σε κάθε κύκλο ρολογιού και να δημιουργήσουμε ένα αντίστοιχο αποτέλεσμα σε κάθε κύκλο ρολογιού μετά από μια αρχική καθυστέρηση.

Θα διαθέσουμε 4 σύγχρονους FA του προηγούμενους ερωτήματος, όπου ο κάθε full adder θα δίνει το αποτέλεσμα του στην έξοδο και το κρατούμενο του στον επόμενο FA. Για την ολοκλήρωση της πρόσθεσης δύο αριθμών των 4 bits θα χρειαστεί να περάσουν 4 κύκλοι, κατά την διάρκεια των οποίων ο κάθε FA θα χρησιμοποιείται διαδοχικά για έναν κύκλο. Αυτό σημαίνει ότι μόλις συμπληρωθεί η πρόσθεση των αντίστοιχων ψηφίων δύο αριθμών από έναν πλήρη αθροιστή, μπορεί να προχωρήσει στον χειρισμό των αντίστοιχων ψηφίων της επόμενης πρόσθεσης. Ωστόσο, για να πραγματοποιηθεί ομαλά αυτή η



Σχήμα 1.13 Ο αθροιστής διάδοσης κρατουμένου σε λειτουργία συνεχούς διοχέτευσης

μεταβίβαση, το κρατούμενο πρέπει να έχει δημιουργηθεί από τον προηγούμενο αθροιστή. Για να διασφαλίσουμε τον σωστό συγχρονισμό θα ενσωματώσουμε καταχωρητές για την αποθήκευση των bit εισόδου A_i και B_i μέχρι να δημιουργηθεί το κρατούμενο C_{i-1}. Επιπλέον, θα προστεθούν καταχωρητές στην έξοδο κάθε πλήρους αθροιστή για να διασφαλιστεί ότι τα αποτελέσματά τουα φτάνουν στην έξοδο ταυτόχρονα. Οι καθυστερήσεις που σχετίζονται με αυτούς τους καταχωρητές υποδεικνύονται με τελείες στο παρακάτω διάγραμμα.

Παραθέτουμε τον κώδικα τόσο για το DFF τόσο και για το αθροιστή διάδοσης κρατουμένου:

1) DFF

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

entity dff is
    Port ( D : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Q : out STD_LOGIC);
end dff;

architecture Behavioral of dff is

begin

    process(clk, rst) begin
        if rst='0' then
            Q<='0';
        elsif rising_edge(clk) then
            Q<=D;
        end if;
    end process;

end Behavioral;
```

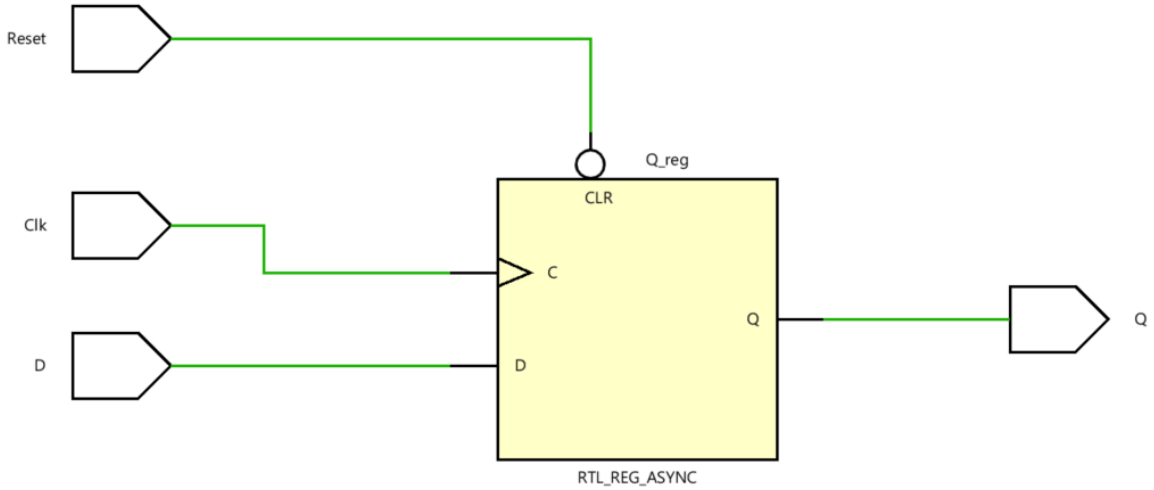
2) Αθροιστής διάδοσης κρατουμένου

```
library IEEE;
```

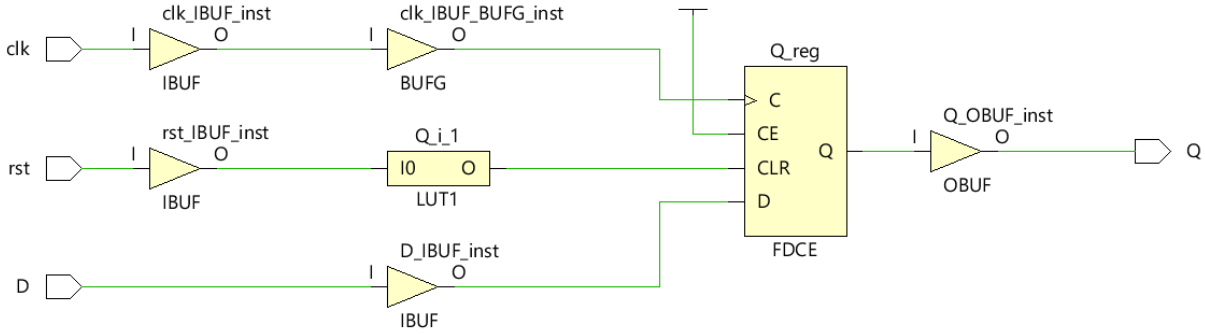
```
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder_4bit_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;
           Sum : out std_logic_vector (3 downto 0);
           Cout : out std_logic);
end full_adder_4bit_pipeline;

architecture Structural of full_adder_4bit_pipeline is

component dff is
    Port ( D : in std_logic;
           clk : in std_logic;
           rst : in std_logic;
           Q : out std_logic);
end component;

component full_adder_bh is
    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;

--signal regCout, regCin : std_logic_vector(2 downto 0);
signal c0, c1, c2, dffA0, dffB0, dffS2 : std_logic;
signal dffA1, dffB1, dffS1 : std_logic_vector(1 downto 0);
signal dffA2, dffB2, dffS0 : std_logic_vector(2 downto 0);

begin

FA0: full_adder_bh port map (a=>A(0), b=>B(0), cin=>Cin, clk=>Clk,
```

```
rst=>Rst, sum=>dffS0(0), cout=>c0);
dffS0_0: dff port map (D=>dffS0(0), clk=>clk, rst=>rst, Q=>dffS0(1));
dffS0_1: dff port map (D=>dffS0(1), clk=>clk, rst=>rst, Q=>dffS0(2));
dffS0_2: dff port map (D=>dffS0(2), clk=>clk, rst=>rst, Q=>Sum(0));

regA0_0: dff port map (D=>A(1), clk=>clk, rst=>rst, Q=>dffA0);
regB0_0: dff port map (D=>B(1), clk=>clk, rst=>rst, Q=>dffB0);
FA1: full_adder_bh port map (a=>dffA0, b=>dffB0, cin=>c0, clk=>Clk,
rst=>Rst, sum=>dffS1(0), cout=>c1);
regS1_0: dff port map (D=>dffS1(0), clk=>clk, rst=>rst, Q=>dffS1(1));
regS1_1: dff port map (D=>dffS1(1), clk=>clk, rst=>rst, Q=>Sum(1));

regA1_0: dff port map (D=>A(2), clk=>clk, rst=>rst, Q=>dffA1(0));
regB1_0: dff port map (D=>B(2), clk=>clk, rst=>rst, Q=>dffB1(0));
regA1_1: dff port map (D=>dffA1(0), clk=>clk, rst=>rst, Q=>dffA1(1));
regB1_1: dff port map (D=>dffB1(0), clk=>clk, rst=>rst, Q=>dffB1(1));
FA2: full_adder_bh port map (a=>dffA1(1), b=>dffB1(1), cin=>c1,
clk=>Clk, rst=>Rst, sum=>dffS2, cout=>c2);
regS2_0: dff port map (D=>dffS2, clk=>clk, rst=>rst, Q=>Sum(2));

regA3_0: dff port map (D=>A(3), clk=>clk, rst=>rst, Q=>dffA2(0));
regB3_0: dff port map (D=>B(3), clk=>clk, rst=>rst, Q=>dffB2(0));
regA3_1: dff port map (D=>dffA2(0), clk=>clk, rst=>rst, Q=>dffA2(1));
regB3_1: dff port map (D=>dffB2(0), clk=>clk, rst=>rst, Q=>dffB2(1));
regA3_2: dff port map (D=>dffA2(1), clk=>clk, rst=>rst, Q=>dffA2(2));
regB3_2: dff port map (D=>dffB2(1), clk=>clk, rst=>rst, Q=>dffB2(2));
FA3: full_adder_bh port map (a=>dffA2(2), b=>dffB2(2), cin=>c2,
clk=>Clk, rst=>Rst, sum=>Sum(3), cout=>Cout);

end Structural;
```

Το RTL Schematic για τον DFF:

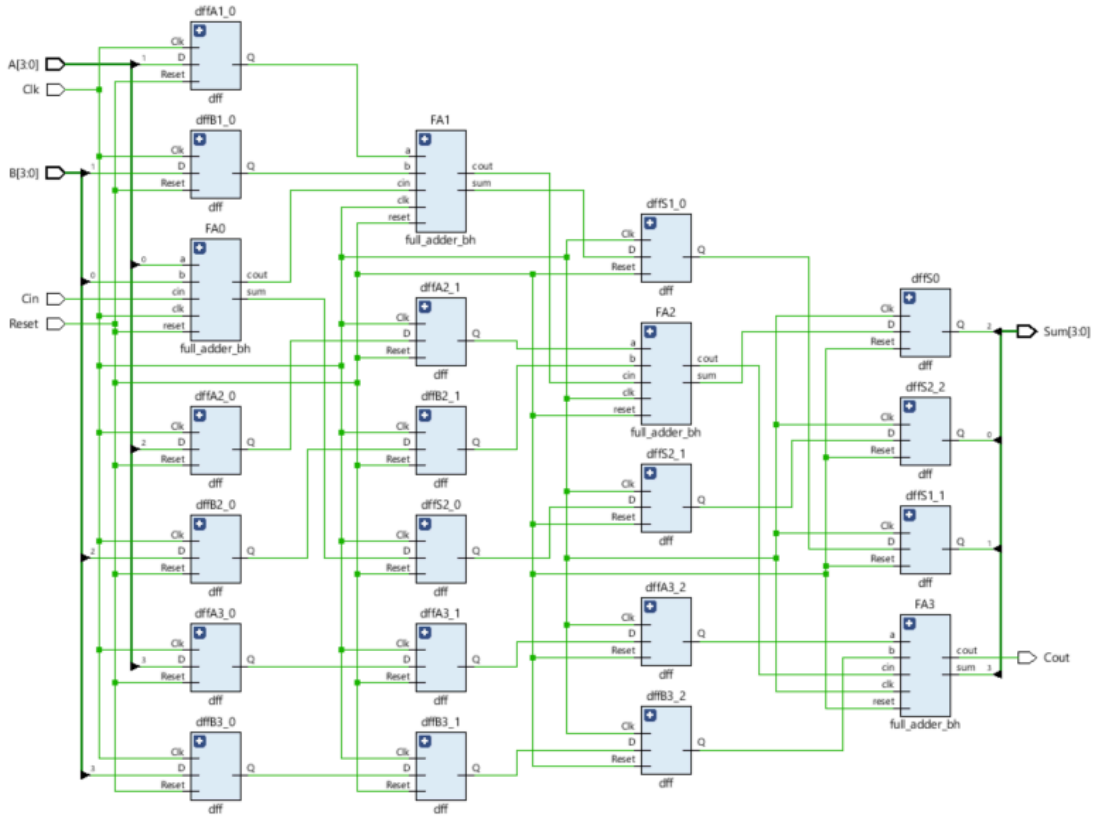


Και το DFF Synthesis Schematic:



Χρησιμοποιώντας Structural Architecture και αξιοποιώντας τις δομές του Θέματος ένα και του dff, καταλήγουμε στο ζητούμενο.

RTL & Synthesis:



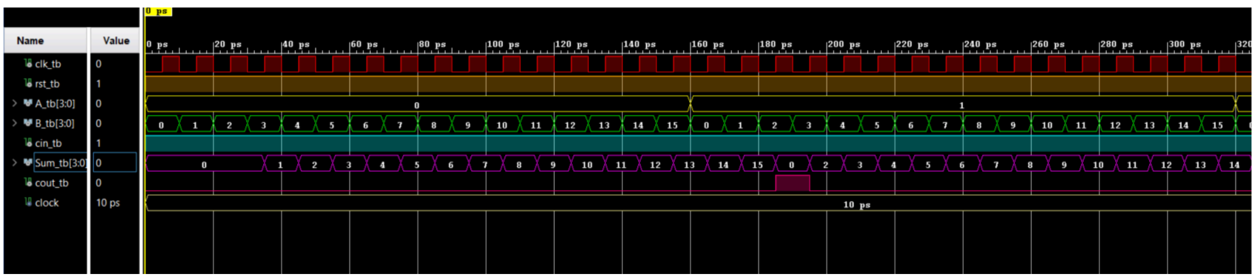
Ο κώδικας για το TestBench φαίνεται παρακάτω:

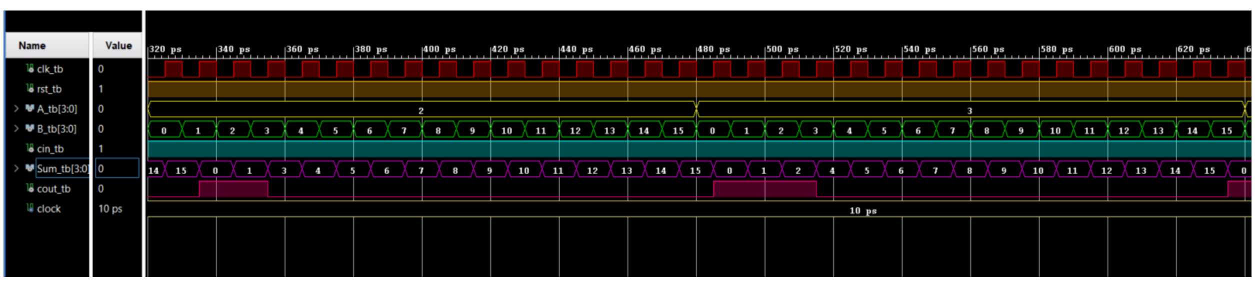
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity full_adder_4bit_pipeline_tb is
--  Port ( );
end full_adder_4bit_pipeline_tb;

architecture bench of full_adder_4bit_pipeline_tb is

    constant clk1_period: time := 270ps;

    component full_adder_4bit_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;
```

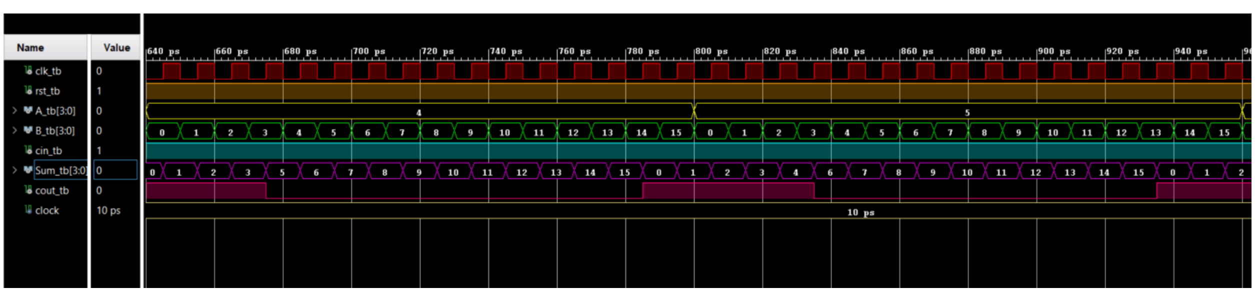
```
            Rst : in std_logic;
            Sum : out std_logic_vector (3 downto 0);
            Cout : out std_logic);
    end component;

    signal clk_tb, rst_tb, cin_tb : std_logic := '0';
    signal cout_tb : std_logic;
    signal A_tb, B_tb : std_logic_vector(3 downto 0) := "1111";
    signal Sum_tb :std_logic_vector(3 downto 0);

    constant clock : time := 10 ps ;

begin
     dut: full_adder_4bit_pipeline
        port map(
            A=>A_tb,
            B=>B_tb,
            Cin => cin_tb,
            Clk=>clk_tb,
            Rst=>rst_tb,
            Sum=>Sum_tb,
            Cout=>cout_tb
            );
    simulation : process
    begin
        for i in 0 to 1 loop
            rst_tb <= not rst_tb;
            for l in 0 to 1 loop
                cin_tb <= not cin_tb;
                for j in 0 to 15 loop
                    A_tb<= A_tb + 1;
                    for k in 0 to 15 loop
                        B_tb <= B_tb + 1;
                        wait for clock;
                    end loop;
                end loop;
            end loop;
        end loop;
        wait;
```

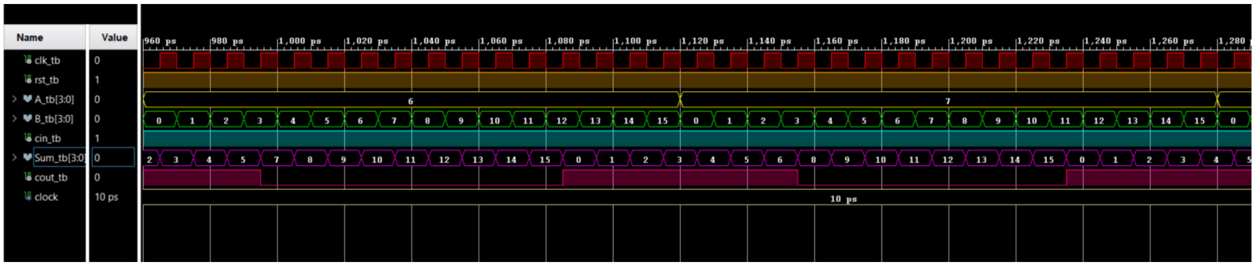
```
    end process;

    generate_clock : process
    begin
            clk_tb <= '0';
            wait for clock/2;
            clk_tb <= '1';
            wait for clock/2;
    end process;
end bench;
```

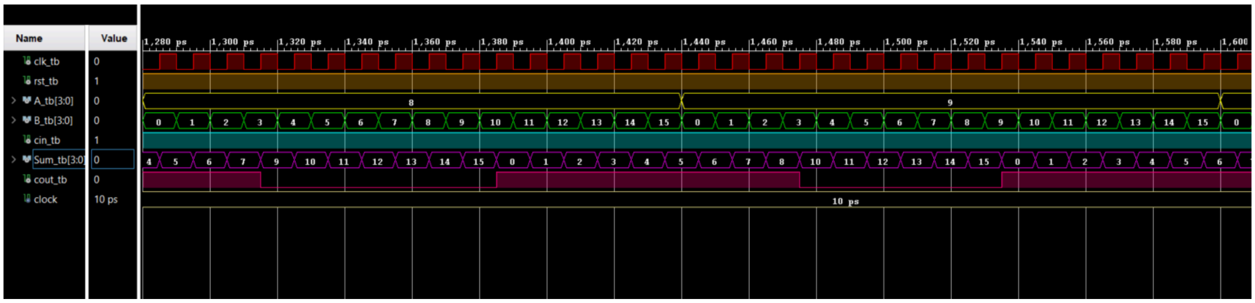
To Simulation:

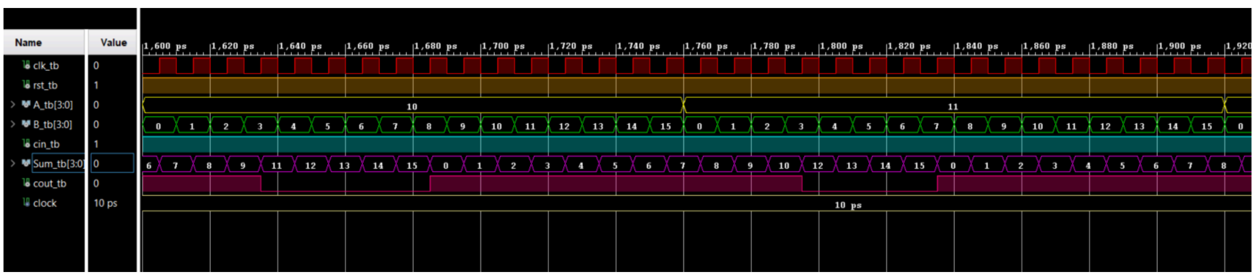


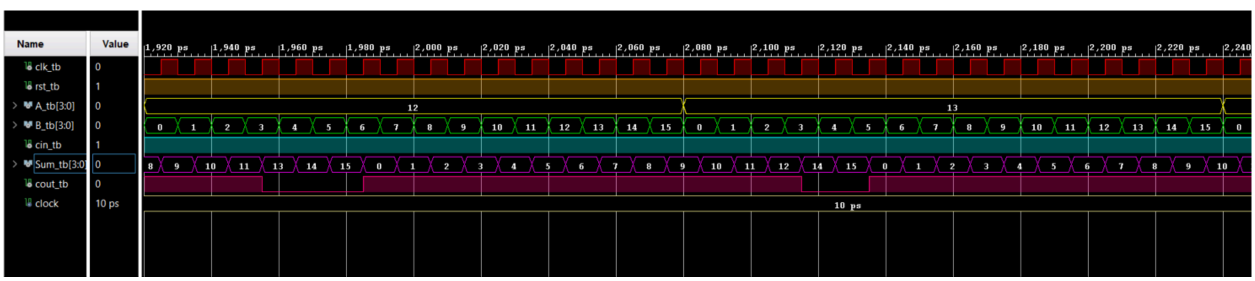


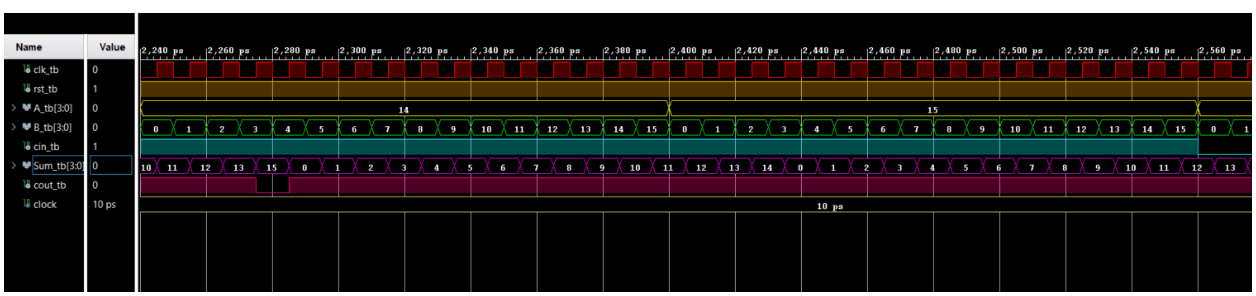












Για το Critical path:

Name	Slack	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	FA3/cout_reg/C	Cout	4.076	3.276	0.800	∞				0.000
Path 2	∞	2	2	1	dffS0_2/Q_reg/C	Sum[0]	4.076	3.276	0.800	∞				0.000
Path 3	∞	2	2	1	regS1_1/Q_reg/C	Sum[1]	4.076	3.276	0.800	∞				0.000
Path 4	∞	2	2	1	regS2_0/Q_reg/C	Sum[2]	4.076	3.276	0.800	∞				0.000
Path 5	∞	2	2	1	FA3/sum_reg/C	Sum[3]	4.076	3.276	0.800	∞				0.000
Path 6	∞	2	3	26	Rst	FA0/cout_reg/CLR	2.748	1.106	1.642	∞	input port clock			0.000
Path 7	∞	2	3	26	Rst	FA0/sum_reg/CLR	2.748	1.106	1.642	∞	input port clock			0.000
Path 8	∞	2	3	26	Rst	FA1/cout_reg/CLR	2.748	1.106	1.642	∞	input port clock			0.000
Path 9	∞	2	3	26	Rst	FA1/sum_reg/CLR	2.748	1.106	1.642	∞	input port clock			0.000
Path 10	∞	2	3	26	Rst	FA2/cout_reg/CLR	2.748	1.106	1.642	∞	input port clock			0.000

Το κρίσιμο μονοπάτι του κυκλώματος είναι για τους 3 πρώτους full adders από το τελευταίο στάδιο καταχωρητών 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 καθυστέρηση του κρίσιμου μονοπατιού είναι 4.076ns.

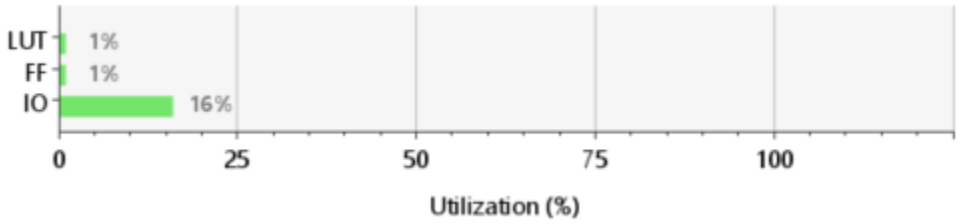
Από την εντολή report_timing_summary -report_unconstrained παίρνουμε πάλι μια πιο αναλυτική αναφορά για το κρίσιμο μονοπάτι.

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack:                    inf
  Source:                 FA3/cout_reg/C
                            (rising edge-triggered cell FDCE)
  Destination:            Cout
                            (output port)
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        4.076ns  (logic 3.276ns (80.380%)  route 0.800ns (19.620%))
  Logic Levels:           2  (FDCE=1 OBUF=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDCE                         0.000     0.000 r  FA3/cout_reg/C
                         FDCE (Prop_fdce_C_Q)         0.456     0.456 r  FA3/cout_reg/Q
                         net (fo=1, unplaced)         0.800     1.256    Cout_OBUF
                         OBUF (Prop_obuf_I_O)         2.820     4.076 r  Cout_OBUF_inst/O
                         net (fo=0)                   0.000     4.076    Cout
                                                                      r  Cout (OUT)
  -------------------------------------------------------------------    -------------------
```

```
Min Delay Paths
--------------------------------------------------------------------------------
Slack:                    inf
  Source:                 FA0/sum_reg/C
                            (rising edge-triggered cell FDCE)
  Destination:            dffS0_0/Q_reg/D
  Path Group:             (none)
  Path Type:              Min at Fast Process Corner
  Data Path Delay:        0.282ns  (logic 0.141ns (50.038%)  route 0.141ns (49.962%))
  Logic Levels:           1  (FDCE=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDCE                         0.000     0.000 r  FA0/sum_reg/C
                         FDCE (Prop_fdce_C_Q)         0.141     0.141 r  FA0/sum_reg/Q
                         net (fo=1, unplaced)         0.141     0.282    dffS0_0/sum
                         FDCE                                         r  dffS0_0/Q_reg/D
  -------------------------------------------------------------------    -------------------
```

Η κατανάλωση πόρων είναι η εξής:

Resource	Utilization	Available	Utilization %
LUT	5	17600	0.03
FF	26	35200	0.07
IO	16	100	16.00



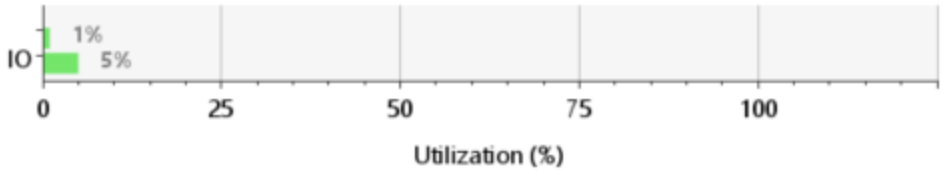
Συγκρίνουμε την παραπάνω υλοποίηση με τον Παράλληλο Αθροιστή της Εργαστηριακής Άσκησης 2. Για τη δομή αυτή, φαίνονται παρακάτω το critical path και το report timing summary:

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	∞	3	4	2	fb	fsum	5.377	3.778	1.599	∞	input port clock			0.000
Path 2	∞	3	4	2	fcin	fcout	5.351	3.752	1.599	∞	input port clock			0.000

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack:                  inf
  Source:               fb
                          (input port)
  Destination:          fsum
                          (output port)
  Path Group:           (none)
  Path Type:            Max at Slow Process Corner
  Data Path Delay:      5.377ns  (logic 3.778ns (70.255%)  route 1.599ns (29.745%))
  Logic Levels:         3  (IBUF=1 LUT3=1 OBUF=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                                                       0.000     0.000 r  fb (IN)
                       net (fo=0)                      0.000     0.000    fb
                       IBUF (Prop_ibuf_I_O)            0.982     0.982 r  fb_IBUF_inst/O
                       net (fo=2, unplaced)            0.800     1.782    fb_IBUF
                       LUT3 (Prop_lut3_I1_O)           0.150     1.932 r  fsum_OBUF_inst_i_1/O
                       net (fo=1, unplaced)            0.800     2.732    fsum_OBUF
                       OBUF (Prop_obuf_I_O)            2.645     5.377 r  fsum_OBUF_inst/O
                       net (fo=0)                      0.000     5.377    fsum
                                                                       r  fsum (OUT)
  -------------------------------------------------------------------    -------------------


Min Delay Paths
--------------------------------------------------------------------------------
Slack:                  inf
  Source:               fa
                          (input port)
  Destination:          fsum
                          (output port)
  Path Group:           (none)
  Path Type:            Min at Fast Process Corner
  Data Path Delay:      2.089ns  (logic 1.415ns (67.729%)  route 0.674ns (32.271%))
  Logic Levels:         3  (IBUF=1 LUT3=1 OBUF=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                                                       0.000     0.000 r  fa (IN)
                       net (fo=0)                      0.000     0.000    fa
                       IBUF (Prop_ibuf_I_O)            0.211     0.211 r  fa_IBUF_inst/O
                       net (fo=2, unplaced)            0.337     0.548    fa_IBUF
                       LUT3 (Prop_lut3_I0_O)           0.042     0.590 r  fsum_OBUF_inst_i_1/O
                       net (fo=1, unplaced)            0.337     0.927    fsum_OBUF
                       OBUF (Prop_obuf_I_O)            1.162     2.089 r  fsum_OBUF_inst/O
                       net (fo=0)                      0.000     2.089    fsum
                                                                       r  fsum (OUT)
  -------------------------------------------------------------------    -------------------
```

Η κατανάλωση πόρων του Πλήρους Αθροιστή είναι η εξής:

Resource	Utilization	Available	Utilization %
LUT	1	17600	0.01
IO	5	100	5.00



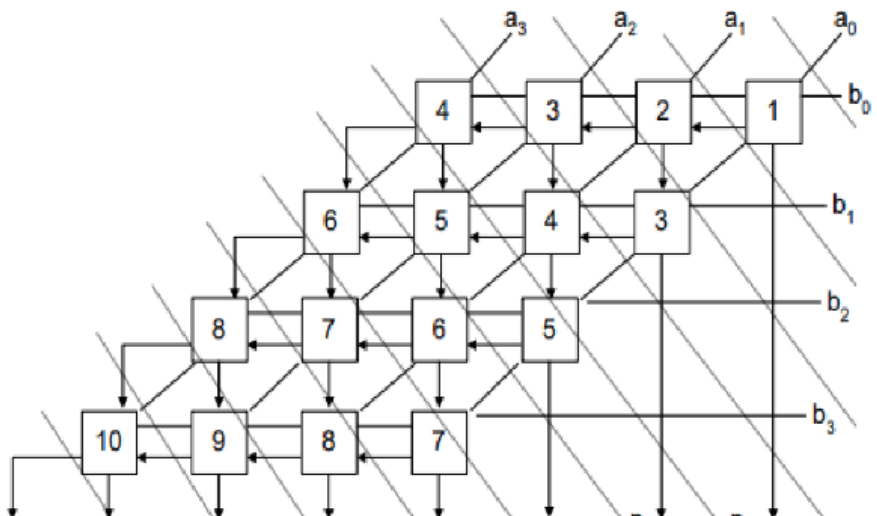
Η περίοδος του παράλληλου αθροιστή με pipeline είναι μικρότερη σε σύγκριση με την έκδοση χωρίς pipeline λόγω της εγγενούς αρχιτεκτονικής του pipeline. Στον non-pipelined αθροιστή, κάθε στάδιο του υπολογισμού πρέπει να ολοκληρωθεί πριν περάσουν τα δεδομένα στο επόμενο στάδιο, προκαλώντας μια καθυστέρηση που παρατείνει τον συνολικό χρόνο επεξεργασίας. Αντίθετα, στον pipelined παράλληλο αθροιστή, το κρίσιμο μονοπάτι περιλαμβάνει κυρίως τη διάδοση δεδομένων μεταξύ καταχωρητών μέσα σε διαδοχικά στάδια, με αποτέλεσμα ένα χαμηλότερο όριο στην περίοδο του ρολογιού. Αυτό συμβαίνει επειδή η pipeline προσέγγιση επιτρέπει την ταυτόχρονη επεξεργασία διαφορετικών σταδίων, μειώνοντας τη συνολική καθυστέρηση.

Θεωρώντας τον παράλληλο αθροιστή με σωλήνωση ως σύγχρονο κύκλωμα, η καθυστέρηση κατά μήκος της κρίσιμης διαδρομής καθορίζει την ελάχιστη περίοδο ρολογιού που απαιτείται για τη σωστή λειτουργία. Αυτή η καθυστέρηση, που συμβαίνει μεταξύ διαδοχικών κύκλων ρολογιού, είναι εγγενώς μικρότερη από τον συνολικό χρόνο υπολογισμού που απαιτείται από τον αθροιστή χωρίς σωλήνωση.

Επιπλέον, η σχεδίαση με σωλήνωση απαιτεί πρόσθετους πόρους υλικού, όπως flip-flops για καταχωρητές και ένα παγκόσμιο buffer ρολογιού. Επιπλέον, η εφαρμογή πλήρους αθροιστές ως διαδοχικά σύγχρονα κυκλώματα και στα δύο σχέδια συμβάλλει στην αυξημένη κατανάλωση πόρων όσον αφορά τα στοιχεία flip-flops, πίνακες αναζήτησης (LUT) και στοιχεία εισόδου/εξόδου (IO). Ως εκ τούτου, ο παράλληλος αθροιστής με pipeline καταναλώνει περισσότερους πόρους λόγω της αρχιτεκτονικής του πολυπλοκότητας και της ανάγκης για πρόσθετα εξαρτήματα για την υποστήριξη του pipelining.

Θέμα 3: Synchronous FA with Pipeline

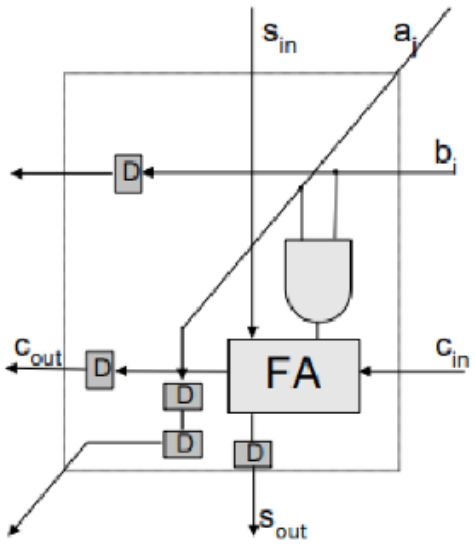
Για τον συστολικό πολλαπλασιαστή διάδοσης κρατουμένων των 4 bits, χρησιμοποιούμε επιπλέον καταχωρητές έτσι ώστε οι είσοδοι για την κάθε δομική μονάδα να φτάνουν ταυτόχρονα στους κατάλληλους κύκλους ρολογιού όπως φαίνεται στο διπλανό σχήμα, ώστε να υπολογίζουμε ορθά το αποτέλεσμα. Επιπλέον, χρησιμοποιούμε καταχωρητές, ώστε όλα τα bit του τελικού



αποτελέσματος να φτάνουν ταυτόχρονα στην έξοδο.

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

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



Κάθε γραμμή του σχήματος αποτελεί τον πολλαπλασιασμό ενός bit του B με τον 4-bit A. Κατά την 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 πολλαπλασιαμό.

Για την δομική μονάδα έχουμε:

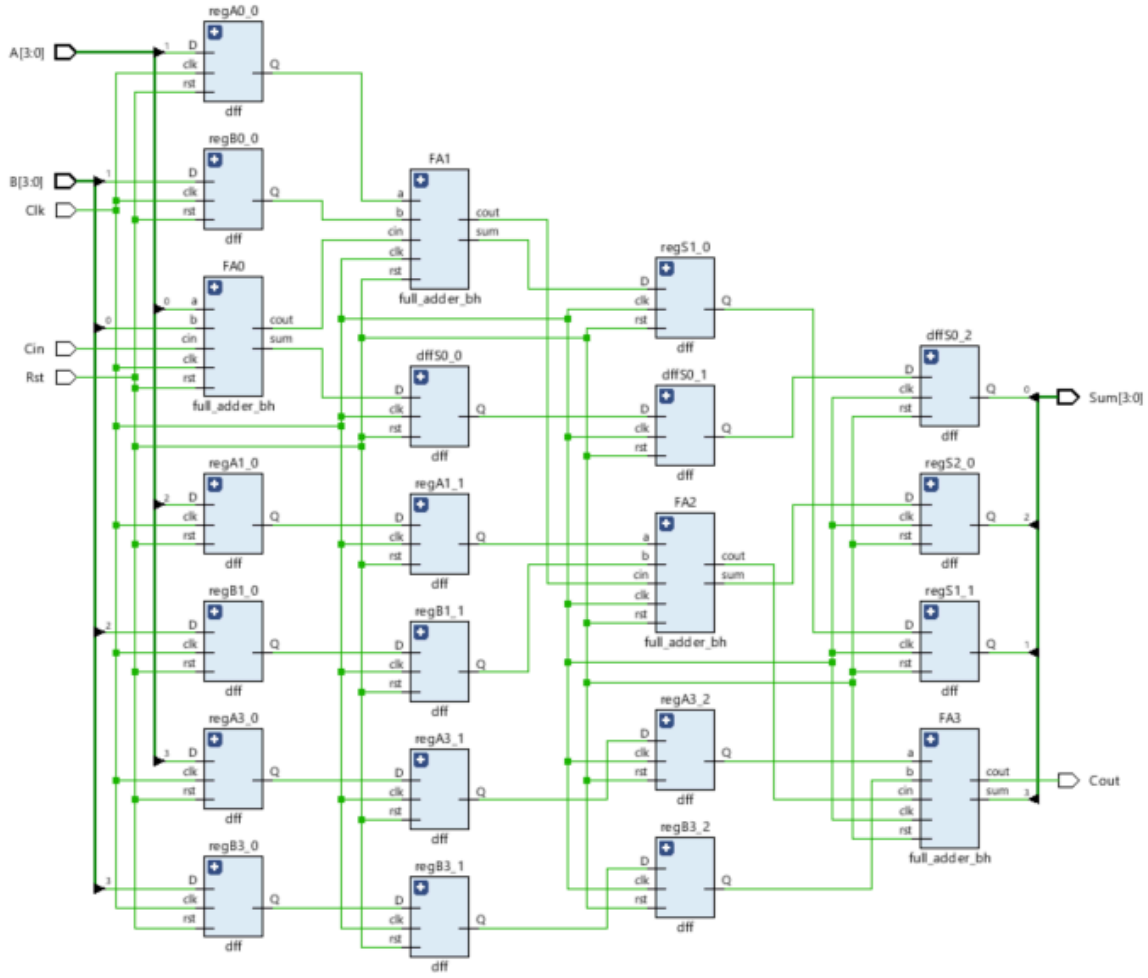
Structural Architecture:

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

entity full_adder_sys_pipeline is
    Port ( a : in std_logic;
           b : in std_logic;
           cin : in std_logic;
           sin : in std_logic;
           clk : in std_logic;
           rst : in std_logic;
           sout : out std_logic;
```

```
            cout : out std_logic;
            D_horizontal : out std_logic;
            D_diagonal : out std_logic);
end full_adder_sys_pipeline;

architecture Structural of full_adder_sys_pipeline is

    component full_adder_bh is
        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;

    component dff is
        Port ( D : in std_logic;
               clk : in std_logic;
               rst : in std_logic;
               Q : out std_logic);
    end component;

    signal dff0, dff1 : std_logic;

begin

    dff0 <= (a and b);
    dff_horizontal : dff port map (D=>b, clk=>clk, rst=>rst,
Q=>D_horizontal);
    dff_diagonal_1 : dff port map (D=>a, clk=>clk, rst=>rst,
Q=>dff1);
    dff_diagonal_2 : dff port map (D=>dff1, clk=>clk, rst=>rst,
Q=>D_diagonal);
    dff_adder : full_adder_bh port map (a=>dff0, b=>sin, cin=>cin,
clk=>clk, rst=>rst, sum=>sout, cout=>cout);
```

```
end Structural;
```

RTL & Synthesis:



Ενώ για τον Ripple carry multiplier:

Structural Architecture:

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

entity mult_4bit_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 mult_4bit_pipeline;

architecture Structural of mult_4bit_pipeline is

    component full_adder_sys_pipeline is
        Port ( a : in std_logic;
               b : in std_logic;
               cin : in std_logic;
               sin : in std_logic;
               clk : in std_logic;
               rst : in std_logic;
               sout : out std_logic;
               cout : out std_logic;
               D_horizontal : out std_logic;
               D_diagonal : out std_logic);
    end component;

    component dff 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 dffP0 : 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 dffP1 : std_logic_vector(6 downto 0);
    signal dffP2 : std_logic_vector(4 downto 0);
    signal dffP3 : std_logic_vector(2 downto 0);
    signal dffP4 : std_logic_vector(1 downto 0);
    signal dffP5 : 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_sys_pipeline port map(a=>A(0), b=>B(0),
cin=>'0', sin=>'0', clk=>clk, rst=>rst, sout=>dffP0(0),
cout=>carry_first_row(0), D_horizontal=>horizontal_first_row(0),
D_diagonal=>diagonal_first_row(0));

dffA1_0: dff port map (D=>A(1), clk=>clk, rst=>rst, Q=>a1_buf);

FIRST_ROW_SECOND: full_adder_sys_pipeline port map(a=>a1_buf,
b=>horizontal_first_row(0), cin=>carry_first_row(0), sin=>'0',
clk=>clk, rst=>rst, sout=>result_first_row(1),
cout=>carry_first_row(1), D_horizontal=>horizontal_first_row(1),
D_diagonal=>diagonal_first_row(1));

dffA2_0: dff port map (D=>A(2), clk=>clk, rst=>rst, Q=>a2_buf(0));
dffA2_1: dff port map (D=>a2_buf(0), clk=>clk, rst=>rst,
Q=>a2_buf(1));

FIRST_ROW_THIRD: full_adder_sys_pipeline port map(a=>a2_buf(1),
b=>horizontal_first_row(1), cin=>carry_first_row(1), sin=>'0',
clk=>clk, rst=>rst, sout=>result_first_row(2),
cout=>carry_first_row(2), D_horizontal=>horizontal_first_row(2),
D_diagonal=>diagonal_first_row(2));
```

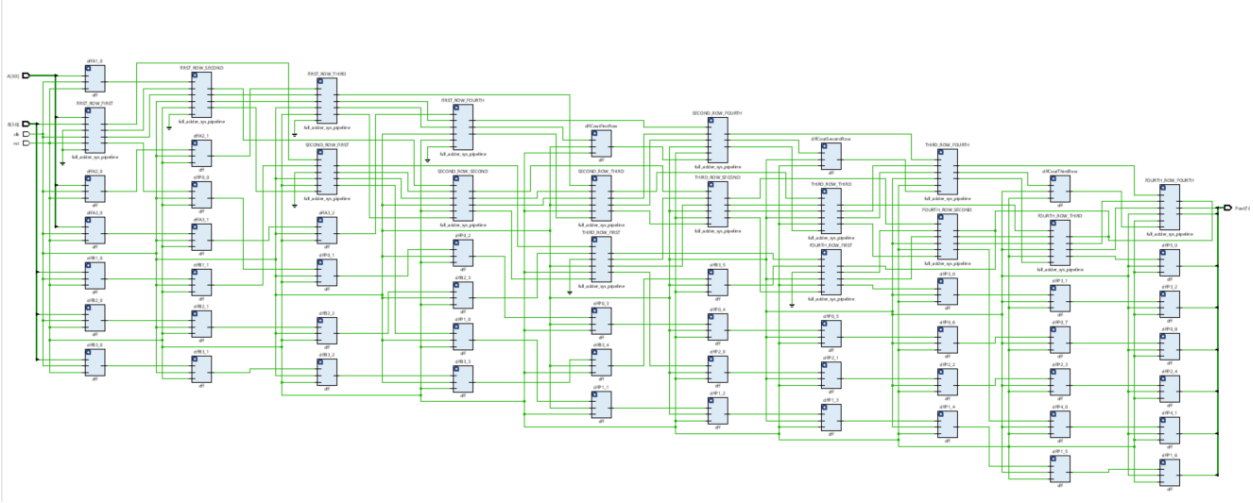
```
dffA3_0: dff port map (D=>A(3), clk=>clk, rst=>rst, Q=>a3_buf(0));
dffA3_1: dff port map (D=>a3_buf(0), clk=>clk, rst=>rst,
Q=>a3_buf(1));
dffA3_2: dff port map (D=>a3_buf(1), clk=>clk, rst=>rst,
Q=>a3_buf(2));

FIRST_ROW_FOURTH: full_adder_sys_pipeline port map(a=>a3_buf(2),
b=>horizontal_first_row(2), cin=>carry_first_row(2), sin=>'0',
clk=>clk, rst=>rst, sout=>result_first_row(3),
cout=>carry_first_row(3), D_horizontal=>horizontal_first_row(3),
D_diagonal=>diagonal_first_row(3));

dffCoutFirstRow: dff port map (D=>carry_first_row(3), clk=>clk,
rst=>rst, Q=>carry_buf_first_row);

dffP0_0: dff port map (D=>dffP0(0), clk=>clk, rst=>rst, Q=>dffP0(1));
dffP0_1: dff port map (D=>dffP0(1), clk=>clk, rst=>rst, Q=>dffP0(2));
dffP0_2: dff port map (D=>dffP0(2), clk=>clk, rst=>rst, Q=>dffP0(3));
dffP0_3: dff port map (D=>dffP0(3), clk=>clk, rst=>rst, Q=>dffP0(4));
dffP0_4: dff port map (D=>dffP0(4), clk=>clk, rst=>rst, Q=>dffP0(5));
dffP0_5: dff port map (D=>dffP0(5), clk=>clk, rst=>rst, Q=>dffP0(6));
dffP0_6: dff port map (D=>dffP0(6), clk=>clk, rst=>rst, Q=>dffP0(7));
dffP0_7: dff port map (D=>dffP0(7), clk=>clk, rst=>rst, Q=>dffP0(8));
dffP0_8: dff port map (D=>dffP0(8), clk=>clk, rst=>rst, Q=>Pout(0));

dffB1_0: dff port map (D=>B(1), clk=>clk, rst=>rst, Q=>b1_buf(0));
dffB1_1: dff port map (D=>b1_buf(0), clk=>clk, rst=>rst,
Q=>b1_buf(1));

SECOND_ROW_FIRST: full_adder_sys_pipeline port
map(a=>diagonal_first_row(0), b=>b1_buf(1), cin=>'0',
sin=>result_first_row(1), clk=>clk, rst=>rst, sout=>dffP1(0),
cout=>carry_second_row(0), D_horizontal=>horizontal_second_row(0),
D_diagonal=>diagonal_second_row(0));
SECOND_ROW_SECOND: full_adder_sys_pipeline port
map(a=>diagonal_first_row(1), b=>horizontal_second_row(0),
cin=>carry_second_row(0), sin=>result_first_row(2), clk=>clk,
rst=>rst, sout=>result_second_row(1), cout=>carry_second_row(1),
```

```
D_horizontal=>horizontal_second_row(1),
D_diagonal=>diagonal_second_row(1));
SECOND_ROW_THIRD: full_adder_sys_pipeline port
map(a=>diagonal_first_row(2), b=>horizontal_second_row(1),
cin=>carry_second_row(1), sin=>result_first_row(3), clk=>clk,
rst=>rst, sout=>result_second_row(2), cout=>carry_second_row(2),
D_horizontal=>horizontal_second_row(2),
D_diagonal=>diagonal_second_row(2));
SECOND_ROW_FOURTH: full_adder_sys_pipeline port
map(a=>diagonal_first_row(3), b=>horizontal_second_row(2),
cin=>carry_second_row(2), sin=>carry_buf_first_row, clk=>clk,
rst=>rst, sout=>result_second_row(3), cout=>carry_second_row(3),
D_horizontal=>horizontal_second_row(3),
D_diagonal=>diagonal_second_row(3));

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

dffP1_0: dff port map (D=>dffP1(0), clk=>clk, rst=>rst, Q=>dffP1(1));
dffP1_1: dff port map (D=>dffP1(1), clk=>clk, rst=>rst, Q=>dffP1(2));
dffP1_2: dff port map (D=>dffP1(2), clk=>clk, rst=>rst, Q=>dffP1(3));
dffP1_3: dff port map (D=>dffP1(3), clk=>clk, rst=>rst, Q=>dffP1(4));
dffP1_4: dff port map (D=>dffP1(4), clk=>clk, rst=>rst, Q=>dffP1(5));
dffP1_5: dff port map (D=>dffP1(5), clk=>clk, rst=>rst, Q=>dffP1(6));
dffP1_6: dff port map (D=>dffP1(6), clk=>clk, rst=>rst, Q=>Pout(1));

dffB2_0: dff port map (D=>B(2), clk=>clk, rst=>rst, Q=>b2_buf(0));
dffB2_1: dff port map (D=>b2_buf(0), clk=>clk, rst=>rst,
Q=>b2_buf(1));
dffB2_2: dff port map (D=>b2_buf(1), clk=>clk, rst=>rst,
Q=>b2_buf(2));
dffB2_3: dff port map (D=>b2_buf(2), clk=>clk, rst=>rst,
Q=>b2_buf(3));

THIRD_ROW_FIRST: full_adder_sys_pipeline port
map(a=>diagonal_second_row(0), b=>b2_buf(3), cin=>'0',
sin=>result_second_row(1), clk=>clk, rst=>rst, sout=>dffP2(0),
cout=>carry_third_row(0), D_horizontal=>horizontal_third_row(0),
D_diagonal=>diagonal_third_row(0));
```

```
THIRD_ROW_SECOND: full_adder_sys_pipeline port
map(a=>diagonal_second_row(1), b=>horizontal_third_row(0),
cin=>carry_third_row(0), sin=>result_second_row(2), clk=>clk,
rst=>rst, sout=>result_third_row(1), cout=>carry_third_row(1),
D_horizontal=>horizontal_third_row(1),
D_diagonal=>diagonal_third_row(1));
THIRD_ROW_THIRD: full_adder_sys_pipeline port
map(a=>diagonal_second_row(2), b=>horizontal_third_row(1),
cin=>carry_third_row(1), sin=>result_second_row(3), clk=>clk,
rst=>rst, sout=>result_third_row(2), cout=>carry_third_row(2),
D_horizontal=>horizontal_third_row(2),
D_diagonal=>diagonal_third_row(2));
THIRD_ROW_FOURTH: full_adder_sys_pipeline port
map(a=>diagonal_second_row(3), b=>horizontal_third_row(2),
cin=>carry_third_row(2), sin=>carry_buf_second_row, clk=>clk,
rst=>rst, sout=>result_third_row(3), cout=>carry_third_row(3),
D_horizontal=>horizontal_third_row(3),
D_diagonal=>diagonal_third_row(3));

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

dffP2_0: dff port map (D=>dffP2(0), clk=>clk, rst=>rst, Q=>dffP2(1));
dffP2_1: dff port map (D=>dffP2(1), clk=>clk, rst=>rst, Q=>dffP2(2));
dffP2_2: dff port map (D=>dffP2(2), clk=>clk, rst=>rst, Q=>dffP2(3));
dffP2_3: dff port map (D=>dffP2(3), clk=>clk, rst=>rst, Q=>dffP2(4));
dffP2_4: dff port map (D=>dffP2(4), clk=>clk, rst=>rst, Q=>Pout(2));

dffB3_0: dff port map (D=>B(3), clk=>clk, rst=>rst, Q=>b3_buf(0));
dffB3_1: dff port map (D=>b3_buf(0), clk=>clk, rst=>rst,
Q=>b3_buf(1));
dffB3_2: dff port map (D=>b3_buf(1), clk=>clk, rst=>rst,
Q=>b3_buf(2));
dffB3_3: dff port map (D=>b3_buf(2), clk=>clk, rst=>rst,
Q=>b3_buf(3));
dffB3_4: dff port map (D=>b3_buf(3), clk=>clk, rst=>rst,
Q=>b3_buf(4));
dffB3_5: dff port map (D=>b3_buf(4), clk=>clk, rst=>rst,
Q=>b3_buf(5));
```

```
FOURTH_ROW_FIRST: full_adder_sys_pipeline port
map(a=>diagonal_third_row(0), b=>b3_buf(5), cin=>'0',
sin=>result_third_row(1), clk=>clk, rst=>rst, sout=>dffP3(0),
cout=>carry_fourth_row(0), D_horizontal=>horizontal_fourth_row(0),
D_diagonal=>dontcare);

dffP3_0: dff port map (D=>dffP3(0), clk=>clk, rst=>rst, Q=>dffP3(1));
dffP3_1: dff port map (D=>dffP3(1), clk=>clk, rst=>rst, Q=>dffP3(2));
dffP3_2: dff port map (D=>dffP3(2), clk=>clk, rst=>rst, Q=>Pout(3));

FOURTH_ROW_SECOND: full_adder_sys_pipeline port
map(a=>diagonal_third_row(1), b=>horizontal_fourth_row(0),
cin=>carry_fourth_row(0), sin=>result_third_row(2), clk=>clk,
rst=>rst, sout=>dffP4(0), cout=>carry_fourth_row(1),
D_horizontal=>horizontal_fourth_row(1), D_diagonal=>dontcare);

dffP4_0: dff port map (D=>dffP4(0), clk=>clk, rst=>rst, Q=>dffP4(1));
dffP4_1: dff port map (D=>dffP4(1), clk=>clk, rst=>rst, Q=>Pout(4));

FOURTH_ROW_THIRD: full_adder_sys_pipeline port
map(a=>diagonal_third_row(2), b=>horizontal_fourth_row(1),
cin=>carry_fourth_row(1), sin=>result_third_row(3), clk=>clk,
rst=>rst, sout=>dffP5, cout=>carry_fourth_row(2),
D_horizontal=>horizontal_fourth_row(2), D_diagonal=>dontcare);

dffP5_0: dff port map (D=>dffP5, clk=>clk, rst=>rst, Q=>Pout(5));

FOURTH_ROW_FOURTH: full_adder_sys_pipeline port
map(a=>diagonal_third_row(3), b=>horizontal_fourth_row(2),
cin=>carry_fourth_row(2), sin=>carry_buf_third_row, clk=>clk,
rst=>rst, sout=>Pout(6), cout=>Pout(7),
D_horizontal=>horizontal_fourth_row(3), D_diagonal=>dontcare);


end Structural;
```

RTL & Synthesis:



Ο κώδικας για το Testbench:

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

entity mult_4bit_pipeline_tb is
-- Port()
end mult_4bit_pipeline_tb;

architecture bench of mult_4bit_pipeline_tb is

    component mult_4bit_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;

    signal A_tb, B_tb : std_logic_vector (3 downto 0) := "1111";
    signal clk_tb, rst_tb : std_logic := '0';
    signal Pout_tb : std_logic_vector (7 downto 0);

    constant clock : time := 10 ps;
```

```
begin
    dut : mult_4bit_pipeline
        port map (
            A=>A_tb,
            B=>B_tb,
            clk=>clk_tb,
            rst=>rst_tb,
            Pout=>Pout_tb
        );

    simulation : process
    begin
        for i in 0 to 1 loop
            rst_tb <= not rst_tb;
            for j in 0 to 15 loop
                A_tb<=A_tb + 1;
                for k in 0 to 15 loop
                    B_tb<=B_tb + 1;
                    wait for clock;
                end loop;
            end loop;
            wait for 10*clock;
        end loop;
    end process;

    generate_clock : process
    begin
        clk_tb <= '0';
        wait for clock/2;
        clk_tb <= '1';
        wait for clock/2;
    end process;

end bench;
```

Simulation:

Η αρχική καθυστέρηση, που ονομάζεται Tlatency, είναι 10 κύκλοι ρολογιού πριν ο πρώτος πολλαπλασιασμός δώσει ένα αποτέλεσμα, το οποίο εκτείνεται επίσης σε 10 κύκλους. Στη συνέχεια, λόγω της εφαρμογής τεχνικής συστολικής σωλήνωσης, κάθε κύκλος ρολογιού παράγει ένα σωστό αποτέλεσμα για τον αντίστοιχο πολλαπλασιασμό. Σε αυτή την τεχνική, το σύστημα χρησιμοποιεί αποτελεσματικά πόρους επεξεργάζοντας δεδομένα με συγχρονισμένο και συνεχή τρόπο σε πολλαπλά στοιχεία επεξεργασίας.

Οι συστολικές συστοιχίες, που συχνά χρησιμοποιούνται σε τέτοιες τεχνικές διοχέτευσης, διευκολύνουν την παράλληλη επεξεργασία δεδομένων μέσω διασυνδεδεμένων στοιχείων επεξεργασίας. Κάθε στοιχείο εκτελεί μια συγκεκριμένη λειτουργία στα δεδομένα και μεταβιβάζει το αποτέλεσμα στο επόμενο στοιχείο, επιτρέποντας μια βελτιωμένη ροή υπολογισμών. Σε αυτό το πλαίσιο, η καθυστέρηση των 10 κύκλων επιτρέπει τη διάδοση των απαραίτητων εισροών μέσω του pipeline, διασφαλίζοντας ότι μέχρι τη στιγμή που το τελευταίο δομικό στοιχείο υπολογίζει το αποτέλεσμα στον 10ο κύκλο, όλες οι απαιτούμενες εισροές έχουν υποστεί σωστή επεξεργασία μέσω των προηγούμενων μπλοκ. Αυτή η συγχρονισμένη προσέγγιση ελαχιστοποιεί τον χρόνο αδράνειας και μεγιστοποιεί τη χρήση των πόρων, με αποτέλεσμα την αποδοτική υπολογιστική απόδοση.

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

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	De
Path 1	∞	2	2	1	dffP0_8/Q_reg/C	Pout[0]	4.076	3.276	0.800	∞		
Path 2	∞	2	2	1	dffP1_6/Q_reg/C	Pout[1]	4.076	3.276	0.800	∞		
Path 3	∞	2	2	1	dffP2_4/Q_reg/C	Pout[2]	4.076	3.276	0.800	∞		
Path 4	∞	2	2	1	dffP3_2/Q_reg/C	Pout[3]	4.076	3.276	0.800	∞		
Path 5	∞	2	2	1	dffP4_1/Q_reg/C	Pout[4]	4.076	3.276	0.800	∞		
Path 6	∞	2	2	1	dffP5_0/Q_reg/C	Pout[5]	4.076	3.276	0.800	∞		
Path 7	∞	2	2	1	FOURTH_ROW...sum_reg/C	Pout[6]	4.076	3.276	0.800	∞		
Path 8	∞	2	2	1	FOURTH_ROW...cout_reg/C	Pout[7]	4.076	3.276	0.800	∞		
Path 9	∞	2	3	88	rst	FIRST_ROW_F..._reg_c/CLR	2.778	1.106	1.672	∞	input port clock	
Path 10	∞	2	3	88	rst	FIRST_ROW_.../Q_reg/CLR	2.778	1.106	1.672	∞	input port clock	

Πιο αναλυτικά το βλέπουμε και εκτελώντας την εντολή report_timing_summary -report_unconstrained :

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack:                    inf
  Source:                 dffP0_8/Q_reg/C
                            (rising edge-triggered cell FDCE)
  Destination:            Pout[0]
                            (output port)
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        4.076ns  (logic 3.276ns (80.380%)  route 0.800ns (19.620%))
  Logic Levels:           2  (FDCE=1 OBUF=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDCE                         0.000     0.000 r  dffP0_8/Q_reg/C
                         FDCE (Prop_fdce_C_Q)         0.456     0.456 r  dffP0_8/Q_reg/Q
                         net (fo=1, unplaced)         0.800     1.256    Pout_OBUF[0]


                         OBUF (Prop_obuf_I_O)         2.820     4.076 r  Pout_OBUF[0]_inst/O
                         net (fo=0)                   0.000     4.076    Pout[0]
                                                                      r  Pout[0] (OUT)
  ---------------------------------------------------------------------    -------------------
```

```
Min Delay Paths
--------------------------------------------------------------------------------------
Slack:                    inf
  Source:                 FIRST_ROW_FIRST/dff_diagonal_1/Q_reg/C
                            (rising edge-triggered cell FDCE)
  Destination:            FIRST_ROW_FIRST/dff_diagonal_2/Q_reg/D
  Path Group:             (none)
  Path Type:              Min at Fast Process Corner
  Data Path Delay:        0.282ns  (logic 0.141ns (50.038%)  route 0.141ns (49.962%))
  Logic Levels:           1  (FDCE=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDCE                         0.000     0.000 r  FIRST_ROW_FIRST/dff_diagonal_1/Q_reg/C
                         FDCE (Prop_fdce_C_Q)         0.141     0.141 r  FIRST_ROW_FIRST/dff_diagonal_1/Q_reg/Q
                         net (fo=1, unplaced)         0.141     0.282    FIRST_ROW_FIRST/dff_diagonal_2/Q
                         FDCE                                        r  FIRST_ROW_FIRST/dff_diagonal_2/Q_reg/D
```