ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (MICROLAB)

3η Εργαστηριακή Αναφορά στο μάθημα "**ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ VLSI "** του 8ου Εξαμήνου

των φοιτητών της **ομάδας 17**,

Εμμανουήλ Αναστάσιου Σερλή, Α.Μ. 03118125
Κωνσταντίνου Ιωάννου, ΑΜ: 03119840

1. Σύγχρονος πλήρης αθροιστής

Κώδικας με περιγραφή συμπεριφοράς (behavioral) :

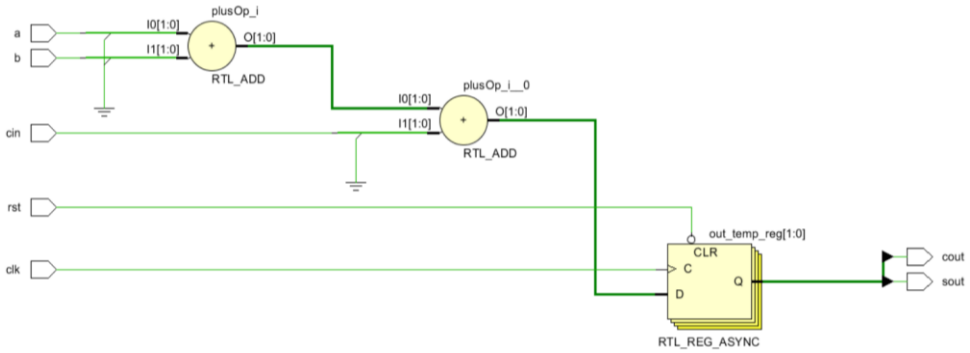
```
entity fa_clk is port(
    a,b,cin,rst,clk: in std_logic;
    sout,cout: out std_logic
);
end fa_clk;

architecture Behavioural of fa_clk is

signal out_temp : std_logic_vector(1 downto 0) := (others => '0');
begin

    process(clk, rst)
    begin
        if rst = '0' then
            out_temp <= "00";
        elsif rising_edge(clk) then
            out_temp <= ('0' & a) + ('0' & b) + ('0' & cin);
        end if;
    end process;
    sout <= out_temp(0);
    cout <= out_temp(1);
end Behavioural;
```

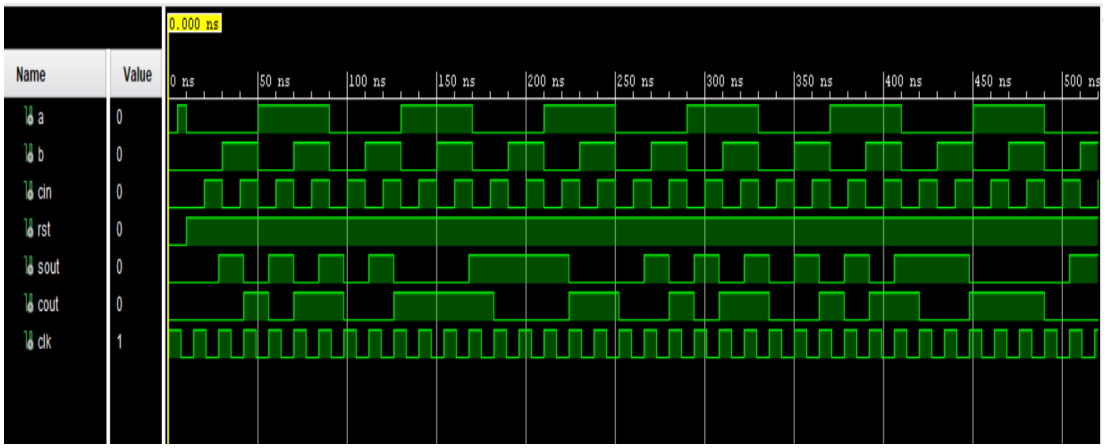
RTL schematic:



Δημιουργούμε ένα testbench με εμφωλευμένα for loops ώστε να συμπεριλάβουμε όλες τις δυνατές εισόδους. Το βασικό κομμάτι του testbench φαίνεται παρακάτω :

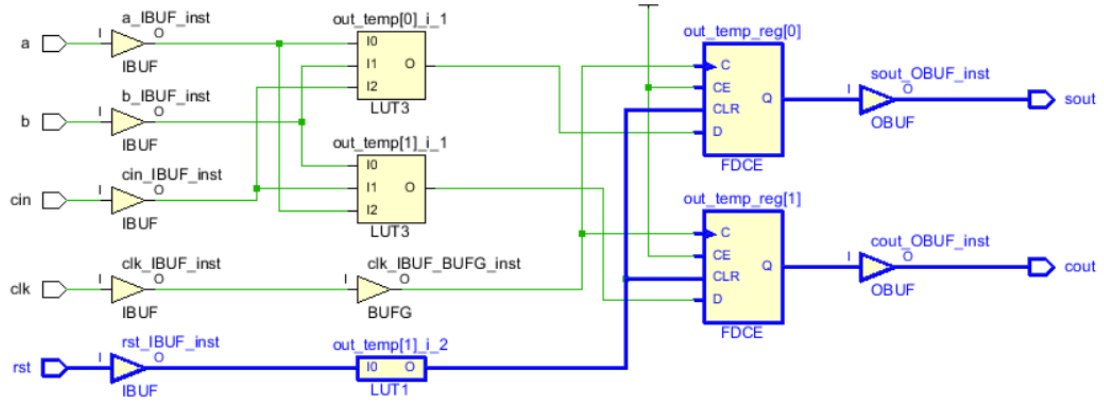
```
for i in 0 to 16 loop
  for i in std_logic range '0' to '1' loop
     a <= i;
      for j in std_logic range '0' to '1' loop
          b <= j;
          for k in std_logic range '0' to '1' loop
              cin <= k;
              wait for 10ns;
```

Το αποτέλεσμα της προσομοίωσης:



Όπως περιμέναμε, σε κάθε θετική ακμή του ρολογιού το κύκλωμα παίρνει τις εισόδους και βγάζει το αναμενόμενο αποτέλεσμα.

Critical Path: **rst->out_temp_reg[1]->Cout** (2.527 + 5.045)ns = 7.572 ns



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	1	1	out_temp_reg[1]/C	cout	5.045	3.237	1.808
Path 2	∞	2	1	1	out_temp_reg[0]/C	sout	4.936	3.075	1.860
Path 3	∞	2	2	2	rst	out_temp_reg[0]/CLR	2.527	1.096	1.430
Path 4	∞	2	2	2	rst	out_temp_reg[1]/CLR	2.527	1.096	1.430
Path 5	∞	2	1	2	a	out_temp_reg[1]/D	2.472	1.102	1.370
Path 6	∞	2	1	2	a	out_temp_reg[0]/D	2.444	1.074	1.370

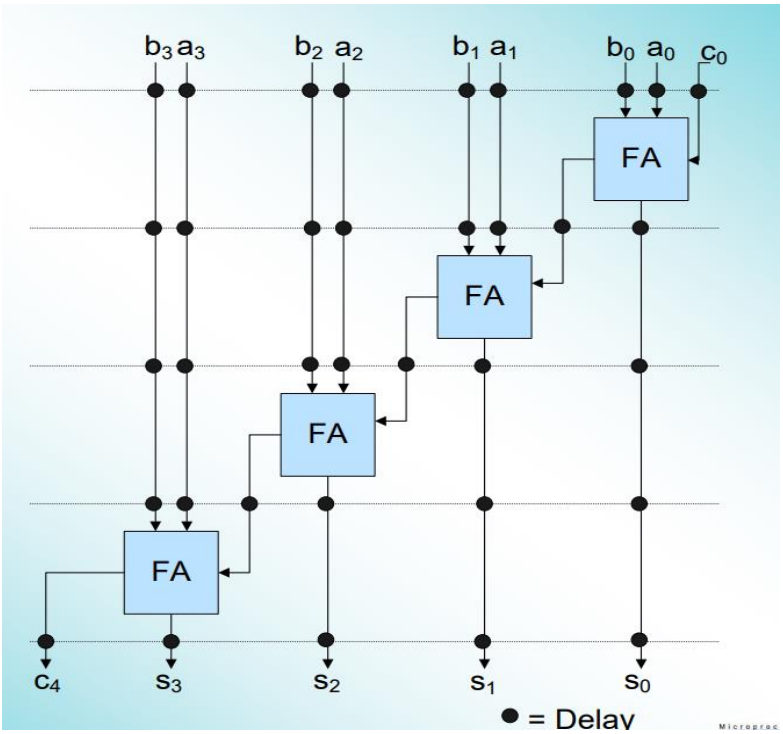
Όπως βλέπουμε από τα σήματα εισόδου το κρίσιμο μονοπάτι αντιστοιχεί στα Paths 3,4 που δηλώνουν τον χρόνο που χρειάζεται το σήμα εισόδου **rst** μέχρι να φτάσει σε κάποιο FCDE **(2.527 ns).**

Στην συνέχεια, το Path 1 είναι το κρίσιμο μονοπάτι από την στιγμή που θα μπουν τα σήματα στο **out_temp_reg[1]** μέχρι να βγουν στην έξοδο ως κρατούμενο, δηλαδή στο **Cout (5.045 ns).**

➔ Τελικά από το rst μέχρι το Cout έχουμε Critical Path με καθυστέρηση **7.572 n**s

2) **Σύγχρονος Αθροιστή διάδοσης κρατουμένου των 4 bits με χρήση της τεχνικής Pipeline**

Χρησιμοποιήσαμε την υλοποίηση που υπάρχει στις διαφάνειες του μαθήματος και το σχηματικό της φαίνεται παρακάτω:



Αρχικά για να δημιουργήσουμε καθυστέρηση παρεμβάλουμε ενδιάμεσα από τα σήματα D flip-flops, τα οποία διατηρούν την πληροφορία εισόδου για έναν ακόμα κύκλο ρολογιού. Έτσι, τα εκάστοτε σήματα καθυστερούν τουλάχιστον ακόμη μια θετική ακμή του ρολογιού για να μπουν ως είσοδοι στους FA ή να βγουν ως έξοδοι από αυτούς.

Κώδικας για DFF:

```
134 entity dff is
135   port(
136        d : in  std_logic;
137        q : out  std_logic;
138        clk : in std_logic;
139        rst : in std_logic
140       );
141 end entity;
142
143 architecture behavioural of dff is
144 begin
145     process(clk, rst)
146     begin
147         if rst = '0' then
148             q <= '0';
149         elsif clk' event and clk = '1' then
150             q <= d;
151         end if;
152     end process;
153 end behavioural;
```

Στην συνέχεια βάζοντας DFF συνδεδεμένα στη σειρά , δημιουργούμε ακόμη τα dff2 και dff3, για να έχουμε 2 και 3 clk delay circuits αντίστοιχα.

```
88
89  entity dff2 is
90      port(
91          d : in  std_logic;
92          q : out  std_logic;
93          clk : in std_logic;
94          rst : in std_logic
95         );
96  end dff2;
97
98  architecture structural of dff2 is
99
00      component dff is...
08
09  signal buffer_bit : std_logic;
10    begin
11      delay3 : dff
12      port map (
13          d => d,
14          q => buffer_bit,
15          clk => clk,
16          rst => rst
17      );
18      delay4 : dff
19      port map (
20          d => buffer_bit,
21          q => q,
22          clk => clk,
23          rst => rst
24      );
```

Όπως φαίνεται για το dff2 χρησιμοποιούμε το dff ως component και συνδέουμε δύο dff σε σειρά, ομοίως εργαζόμαστε και για το dff3.

Κώδικας για την υλοποίηση του ερωτήματος:

```
157 entity fa_pipe is
158   port(
159        a : in  std_logic_vector(3 downto 0);
160        b : in  std_logic_vector(3 downto 0);
161        cin : in  std_logic;
162        clk : in std_logic;
163        rst : in std_logic;
164        s : out std_logic_vector(3 downto 0);
165        cout : out std_logic
166       );
167 end fa_pipe;
168
169 architecture structural of fa_pipe is
170
171    component fa_clk is...
182
183    component dff is...
191
192    component dff2 is...
200
201   component dff3 is...
209
210    signal s_out : std_logic_vector(3 downto 0) := (others => '0');
211   signal car_temp : std_logic_vector(4 downto 0) := (others => '0');
212   signal s_temp : std_logic_vector(3 downto 0) := (others => '0');
213   signal a_temp : std_logic_vector(3 downto 0) := (others => '0');
214   signal b_temp : std_logic_vector(3 downto 0) := (others => '0');
215
```

Αρχικά ορίζουμε τις εισόδους και τις εξόδους του κυκλώματος. Θα χρησιμοποιήσουμε 4 συγχρόνους FAs του προηγούμενου ερωτήματος για να σχεδιάσουμε τον 4 bit FA και τα DFFs για να συμπεριλάβουμε τις κατάλληλες καθυστερήσεις.

Στην συνέχεια συνδέουμε κατάλληλα τους 4 αθροιστές μεταξύ τους:

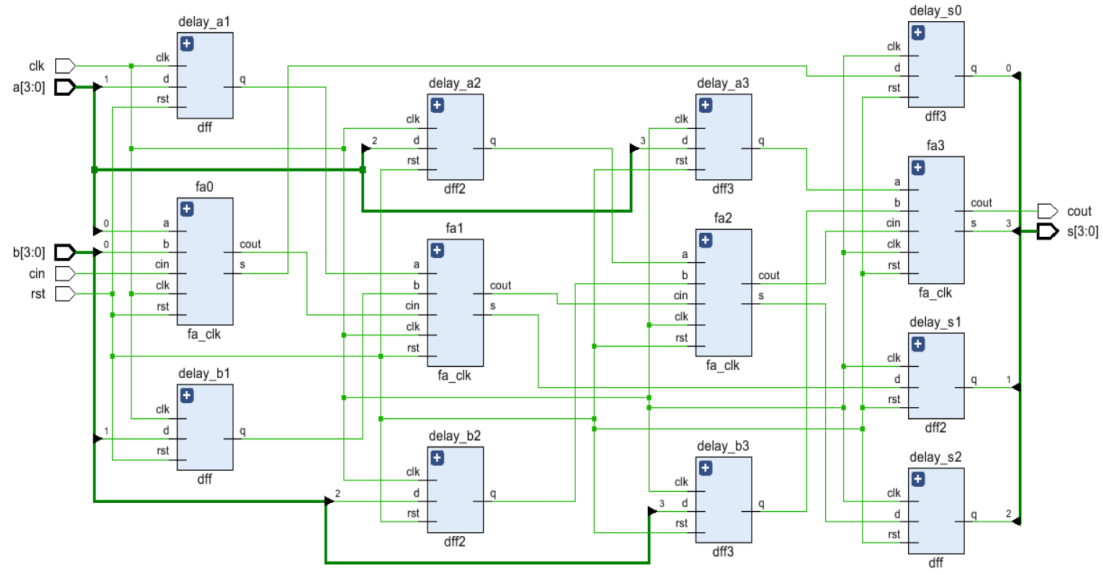
```
215
216   begin
217
218     a_temp(0) <= a(0);
219     b_temp(0) <= b(0);
220     car_temp(0) <= cin;
221
222
223   fa0: fa_clk
224     port map (
225          a => a_temp(0),
226          b => b_temp(0),
227          cin => car_temp(0),
228          cout => car_temp(1),
229          clk => clk,
230          rst => rst,
231          s => s_temp(0)
232          );
233
234   fa1: fa_clk
235     port map (
236          a => a_temp(1),
237          b => b_temp(1),
238          cin => car_temp(1),
239          cout => car_temp(2),
240          clk => clk,
241          rst => rst,
242          s => s_temp(1)
243          );
244
```

```
244
245   fa2: fa_clk
246     port map (
247          a => a_temp(2),
248          b => b_temp(2),
249          cin => car_temp(2),
250          cout => car_temp(3),
251          clk => clk,
252          rst => rst,
253          s => s_temp(2)
254          );
255
256   fa3: fa_clk
257     port map (
258          a => a_temp(3),
259          b => b_temp(3),
260          cin => car_temp(3),
261          cout => car_temp(4),
262          clk => clk,
263          rst => rst,
264          s => s_temp(3)
265          );
266
```

Τέλος, βάζουμε τα flipflops ώστε να υπάρχουν οι σωστές καθυστερήσεις τόσο στα σήματα εισόδου όσο και στο σήμα εξόδου.

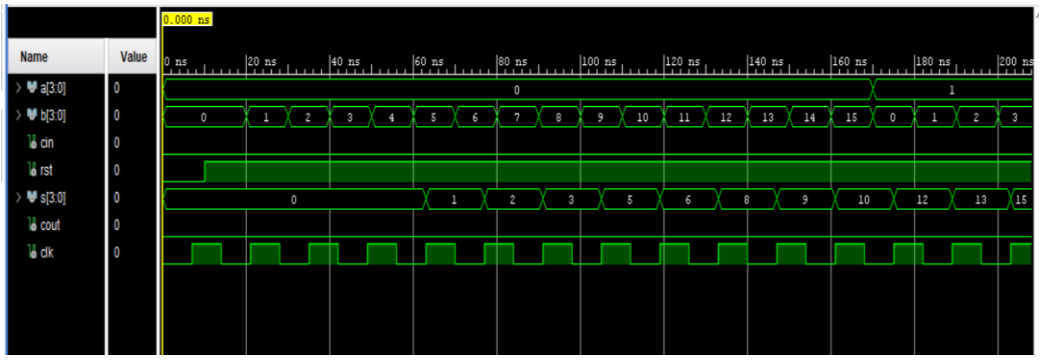
```
268     delay_s0 : dff3...
275
276     delay_s1 : dff2...
283
284     delay_s2 : dff...
291
292     delay_a1 : dff...
299
300      delay_a2 : dff2...
307
308      delay_a3 : dff3...
315
316      delay_b1 : dff...
323
324      delay_b2 : dff2...
331
332      delay_b3 : dff3...
339
340        s_out(3) <= s_temp(3)
341        s <= s_out;
342        cout <= car_temp(4);
343  end architecture ; -- arc
```

RTL schematic:

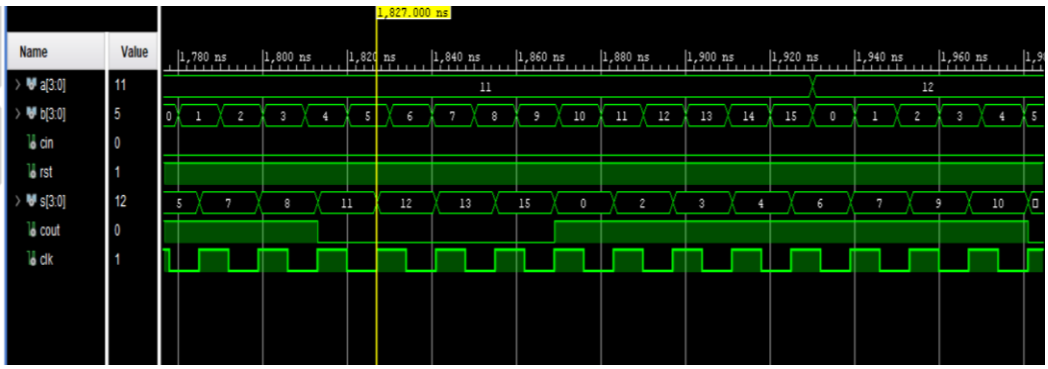


Γράφοντας κατάλληλο testbench έχουμε την κάτωθι **προσομοίωση** :

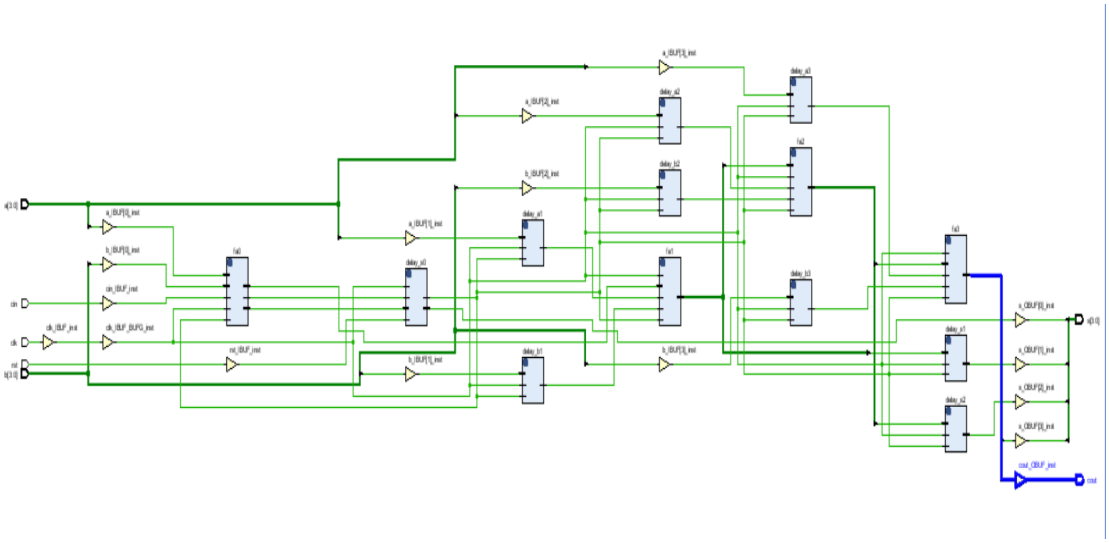
Βλέπουμε ότι σε κάθε θετικό παλμό του ρολογιού παίρνουμε τις εισόδους αυτήν την χρονική στιγμή και μετά από 3 κύκλους ρολογιού βγάζουμε το αποτέλεσμα του αθροίσματός τους στα σήματα s και cout. Συνεπώς, αυτό αντιστοιχεί σε **Tlatency = 3 κύκλοι ρολογιού.**



Επίσης σημειώνουμε ότι όταν το άθροισμα έχει τιμή μεγαλύτερη του 16 τότε το cout γίνεται 1 ,δηλαδή το κρατούμενο αναπαριστά μια 16αδα. Για παράδειγμα, αν έχουμε α=11 και b=5, τότε μετά από 3 κύκλους ρολογιού έχουμε στην έξοδο Sum=0 και Cout =1.



Critical Path: **Input -> Cout 5.332ns**

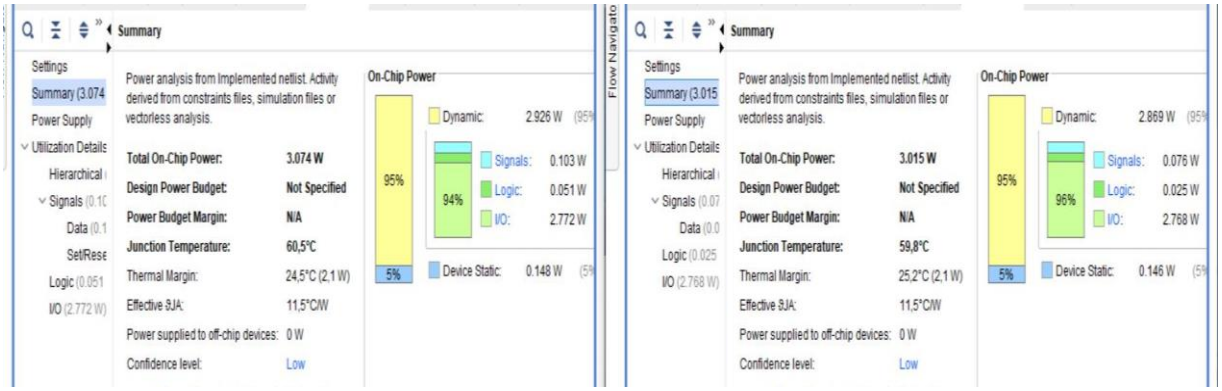


Path 1	∞	2	1	1	fa3/out_temp_reg[1]/C	cout	5.332	3.269	2.064
Path 2	∞	2	1	1	fa3/out_temp_reg[0]/C	s[3]	5.013	3.142	1.871
Path 3	∞	2	1	1	delay_s2/q_reg/C	s[2]	4.973	3.117	1.856
Path 4	∞	2	1	1	delay_s1/delay4/q_reg/C	s[1]	4.906	3.189	1.718

Παρατηρούμε λοιπόν ότι το κρίσιμο μονοπάτι είναι από την στιγμή που θα δοθεί η είσοδος στον 4bit FA μέχρι να βγει στο Cout είναι **5.332 ns**. Αρκετά μικρότερη καθυστέρηση σε σχέση με τον απλό 4-bit FA που σχεδιάσαμε στην 2η Εργαστηριακή Άσκησή, του οποίου το delay ήταν **7.96ns**. Οπότε το Pipeline μείωσε σημαντικά το κρίσιμο μονοπάτι. Το μόνο μειονέκτημα είναι ότι έχουμε μια αρχική καθυστέρηση των τριών κύκλων ρολογιού**.**

Τώρα θα συγκρίνουμε την κατανάλωση πόρων 4bit FA με και χωρίς pipeline :

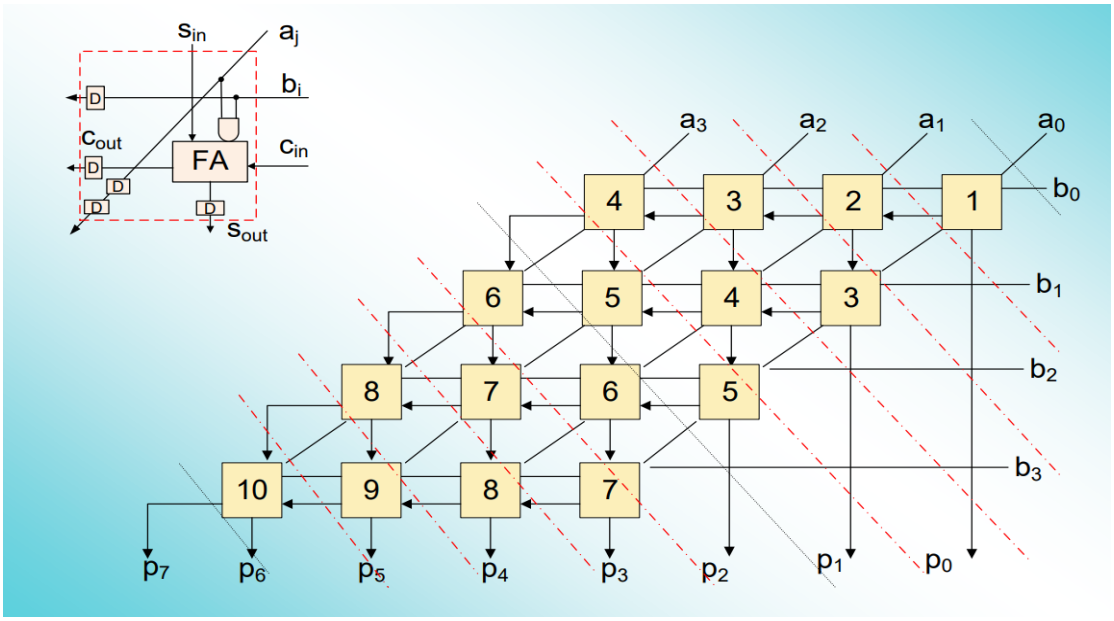
Με pipeline **Χωρίς Pipeline**



Βλέπουμε λοιπόν ότι η σχεδίαση που χρησιμοποιεί pipeline έχει μεγαλύτερη συνολική κατανάλωση από ότι η υλοποίηση χωρίς pipeline. Κάτι τέτοιο είναι λογικό αν σκεφτούμε ότι χρησιμοποιούμε επιπλέον κάποια DFF για να συγχρονίσουμε το pipeline.

3**) Συστολικός (είδος pipeline) Πολλαπλασιαστής διάδοσης κρατουμένων των 4 bit κάνοντας χρήση σύγχρονων Πλήρων Αθροιστών (Full Adders)**

Βασιζόμαστε και πάλι στην υλοποίηση που υπάρχει στις διαφάνειες του μαθήματος



Αρχικά τροποποιούμε τον σύγχρονο FA του ερωτήματος ένα ώστε να προσθέσουμε κατάλληλες καθυστερήσεις στα ai,bi όπως φαίνεται στο παραπάνω σχήμα. Σημειώνουμε ότι το Sout,Cout δεν θα χρειαστούν flip-flop επιπλέον γιατί καθώς ο FA είναι σύγχρονος λειτουργεί ήδη με ένα dff καθυστέρηση.

Υλοποίηση του νέου σύγχρονου FA*:

```
172   signal out_temp : std_logic_vector(1 downto 0) := (others => '0');
173   signal gate : std_logic;
174   signal w1,w2,w3,w4,w5,w6 :std_logic;
175
176   begin
177       gate <= A and B;
178       process(clk, rst)
179       begin
180
181           if rst = '0' then
182               out_temp <= "00";
183           elsif clk' event and clk = '1' then
184               out_temp <= ('0' & gate) + ('0' & Sin) + ('0'&C_in);
185           end if;
186       end process;
187
188       S <= out_temp(0);
189     C_out <= out_temp(1);
190
191     --  B_next
192           delay_B : dff...
199
200      --  A_next
201           delay_A : dff2...
208
209   end behavioural;
```

Κώδικας για Σύγχρονο Multiplier:

```
entity multiplier is
    port (
        A, B : in  std_logic_vector(3 downto 0);
        clk,rst: in std_logic;
        P    : out std_logic_vector(7 downto 0)
    );
end multiplier;

architecture behavioral of multiplier is

    component fa is...

    component dff is...

        component dff2 is...

    component dff3 is...

    signal  S1, S2, S3,S5,S6,S7,S9,S10,S11,C1, C2, C3, C4, C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15 : std_logic;
    signal A_temp, B_temp: std_logic_vector(3 downto 0);
    signal P_temp: std_logic_vector(7 downto 0);
    signal B21, B31, B32: std_logic;
    signal A_next,B_next:  std_logic_vector(15 downto 0);
    signal w1,w2,w3,w4,w5,w6,w7 : std_logic;
    signal temp_C4,temp_C8,temp_C12: std_logic;
```

Στην συνέχεια τοποθετούμε τους 16 FAs* όπως φαίνονται στο αρχικό σχήμα με κατάλληλες εισόδους και εξόδους.

```
begin
   -- First row of FA components
   fa_0 : fa port map (clk=>clk,rst=>rst,Sin=> '0',A=>A_temp(0),B=> B_temp(0),C_in  => '0', S=> P_temp(0), C_out => C1,A_next => A_next(0),B_next =>B_next(0));
   fa_1 : fa port map (clk=>clk,rst=>rst,Sin=> '0',A=>A_temp(1),B=> B_next(0),C_in=> C1,    S=> S1  ,C_out=> C2  ,A_next => A_next(1),B_next =>B_next(1));
   fa_2 : fa port map (clk=>clk,rst=>rst,Sin=> '0',A=>A_temp(2),B=> B_next(1),C_in=> C2,    S=> S2   ,C_out=> C3  ,A_next => A_next(2),B_next =>B_next(2));
   fa_3 : fa port map (clk=>clk,rst=>rst,Sin=> '0',A=>A_temp(3),B=> B_next(2),C_in=> C3,    S=> S3  ,C_out=> temp_C4 ,A_next => A_next(3),B_next =>B_next(3));

   -- Second row of FA components
   fa_4 : fa port map (clk=>clk,rst=>rst,Sin=>S1,A=>A_next(0),B=> B_temp(1),C_in=> '0',S=> P_temp(1),C_out=> C5,A_next => A_next(4),B_next =>B_next(4));
   fa_5 : fa port map (clk=>clk,rst=>rst,Sin=>S2,A=>A_next(1),B=> B_next(4),C_in=> C5, S=> S5,  C_out=> C6,A_next => A_next(5),B_next =>B_next(5));
   fa_6 : fa port map (clk=>clk,rst=>rst,Sin=>S3,A=>A_next(2),B=> B_next(5),C_in=> C6, S=> S6,  C_out=> C7,A_next => A_next(6),B_next =>B_next(6));
   fa_7 : fa port map (clk=>clk,rst=>rst,Sin=>C4,A=>A_next(3),B=> B_next(6),C_in=> C7, S=> S7,  C_out=> temp_C8,A_next => A_next(7),B_next =>B_next(7));

   -- Third row of FA components
    fa_8 : fa port map  (clk=>clk,rst=>rst,Sin=>S5,A=>A_next(4), B=> B_temp(2),C_in=> '0',S=> P_temp(2),C_out=> C9,A_next => A_next(8),B_next =>B_next(8));
    fa_9 : fa port map  (clk=>clk,rst=>rst,Sin=>S6,A=>A_next(5), B=> B_next(8),C_in=> C9, S=> S9,  C_out=> C10,A_next => A_next(9),B_next =>B_next(9));
    fa_10 : fa port map (clk=>clk,rst=>rst,Sin=>S7,A=>A_next(6), B=> B_next(9),C_in=> C10,S=> S10, C_out=> C11,A_next => A_next(10),B_next =>B_next(10));
    fa_11 : fa port map (clk=>clk,rst=>rst,Sin=>C8,A=>A_next(7), B=> B_next(10),C_in=> C11, S=> S11, C_out=> temp_C12,A_next => A_next(11),B_next =>B_next(11));

   -- Fourth row of FA components
   fa_12 : fa port map  (clk=>clk,rst=>rst,Sin=>S9,A=>A_next(8), B=> B_temp(3),C_in=> '0',S=> P_temp(3), C_out=> C13, A_next => A_next(12),B_next =>B_next(12));
   fa_13 : fa port map  (clk=>clk,rst=>rst,Sin=>S10,A=>A_next(9),B=> B_next(12),C_in=> C13,S=> P_temp(4), C_out=> C14, A_next => A_next(13),B_next =>B_next(13));
   fa_14 : fa port map  (clk=>clk,rst=>rst,Sin=>S11,A=>A_next(10),B=> B_next(13),C_in=> C14,S=> P_temp(5), C_out=> C15,A_next => A_next(14),B_next =>B_next(14));
   fa_15 : fa port map  (clk=>clk,rst=>rst,Sin=>C12,A=>A_next(11),B=> B_next(14),C_in=> C15,S=> P_temp(6), C_out=> P_temp(7),A_next => A_next(15),B_next =>B_next(15));
```

Εύκολα παρατηρεί κανείς ότι αποθηκεύουμε τις εξόδους του mul σε προσωρινά σήματα P_temp, όπως και τα ακριανά κρατούμενα κάθε γραμμής temp_C, ενώ το ίδιο κάνουμε και για τις εισόδους A_temp, B_temp. Αυτό το κάνουμε προκειμένου να εισαγάγουμε κάποιες επιπλέον καθυστερήσεις σε αυτά τα σήματα ώστε να φτάνουν όλες οι είσοδοι ταυτόχρονα σε κάθε FA (δηλαδή μαζί με το κρατούμενο του προηγουμένου FA) καθώς και για να βγαίνει το αποτέλεσμα στην έξοδο ταυτόχρονα δηλαδή τα P0 εως P6 χρειάζεται να καθυστερήσουν ώστε να βγεί το P7.

```
294     --Διαγωνίοι delay  1 dff στον καθένα
295     delay_C4: dff...
302
303     delay_C8: dff...
310
311     delay_C12: dff...
318
319
320       --A delay stuff
321       A_temp(0) <= A(0);
322       delay_a1_1: dff...
329       delay_a2_2: dff2...
336       delay_a3_3: dff3...
343
344        --B_stuff
345          B_temp(0) <= B(0);
346
347       --B(1) 2_dff
348       delay_b1_2: dff2...
355
356       -- B(2) 4_dff
357       delay_b2_4_vol1: dff2...
364       delay_b2_4_vol2: dff2...
371
372        -- B(3) 6_dff
373       delay_b3_6_vol1: dff2...
380       delay_b3_6_vol2: dff2...
387       delay_b3_6_vol3: dff2...
394
```

Τα διαγώνια κρατούμενα χρειάζονται έναν επιπλέον κύκλο καθυστέρηση ώστε να φτάσουν μαζί με τα υπόλοιπα σήματα στους FAs.

Το A(0) δεν χρειάζεται καθυστέρηση
Το A(1) χρειάζεται 1 delay, το A(2) 2 delays , το A(3) 3 delays, ώστε να περιμένουν να ετοιμαστούν τα κρατούμενα των προηγούμενων σταδίων.

Με την ίδια λογική εισαγάγουμε τις καθυστερήσεις στα B(0) – B(3) ώστε να συγχρονίζονται με το Sout της παραπάνω γραμμής αθροιστών που το δέχονται ως είσοδο.

Τέλος έχουμε τις καθυστερήσεις στις εξόδους:

```
395   --P_temp
396   P(6) <= P_temp(6);
397   P(7) <= P_temp(7);
398
399   delay_P5 : dff...
406
407   delay_P4 : dff2...
414
415
416   delay_P3 : dff3...
423
424
425   delay_P2_A : dff3...
432   delay_P2_B : dff2...
439
440
441   delay_P1_A : dff3...
448   delay_P1_B : dff3...
455   delay_P1_C : dff...
462
463   delay_P0_A : dff3...
470   delay_P0_B : dff3...
477   delay_P0_C : dff3...
484
485   end behavioral;
486
```

Ομοίως όλες οι έξοδοι χρειάζεται να περιμένουν τα P6, P7 που χρειάζονται:

9 delays σε σχέση με το P(0)

7 delays σε σχέση με το P(1)

5 delays σε σχέση με το P(2)

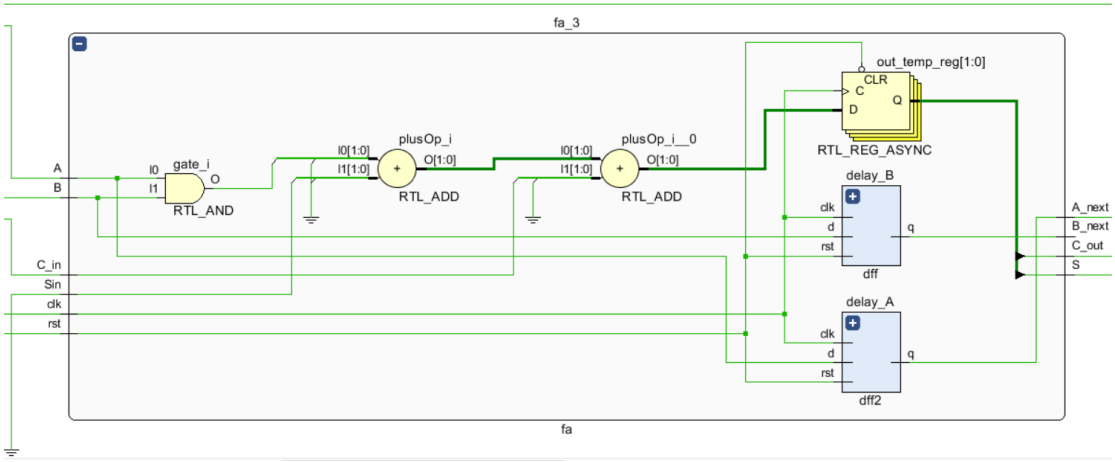
3 delays σε σχέση με το P(3)

2 delays σε σχέση με το P(4)

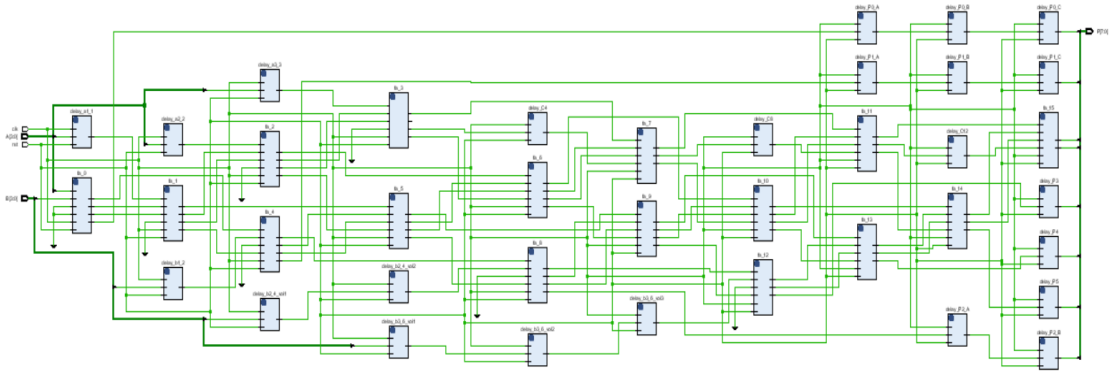
1 delays σε σχέση με το P(1)

RTL Schematics:

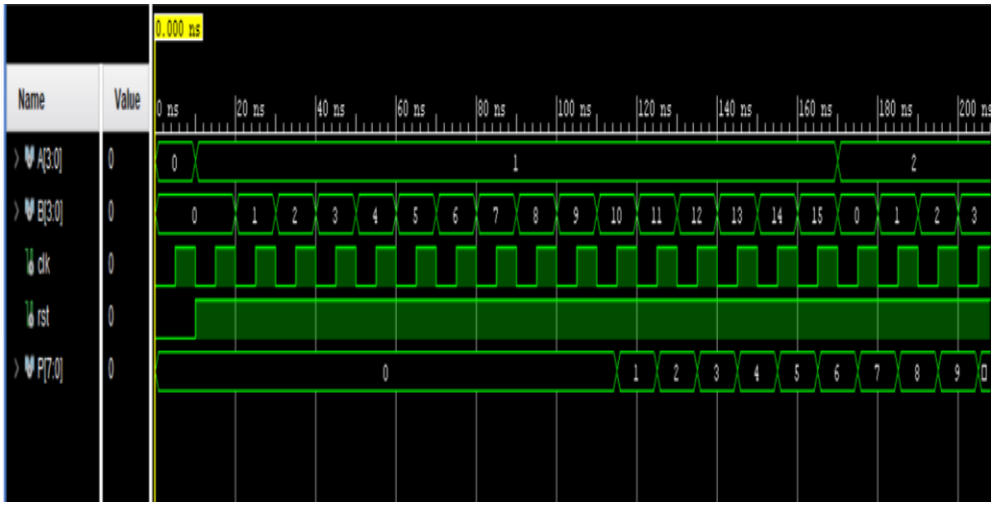
The new FA*:



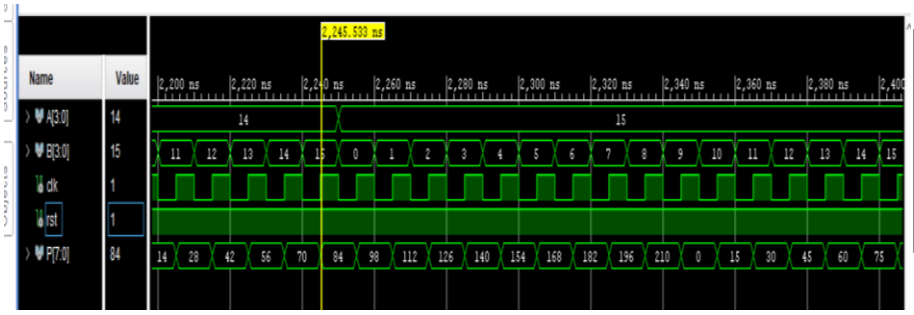
The MUL:



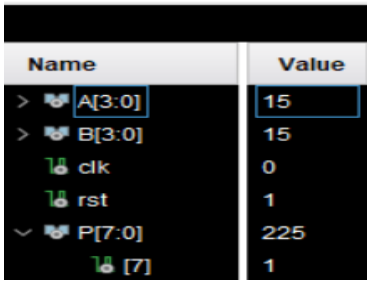
Γράφοντας κατάλληλο testbench έχουμε τα κάτωθι αποτελέσματα από την προσομοίωση:



Παρατηρούμε ότι ο πολλαπλασιαστής δουλεύει κανονικά, μιας και παίρνει σε κάθε ακμή του ρολογιού τις 4-bit εισόδους και τις πολλαπλασιάζει. Το αποτέλεσμα του πολλαπλασιασμού είναι έτοιμο μετά από καθυστέρηση 9 κύκλων ρολογιού δηλαδή έχουμε **Tlatency = 9clks**.

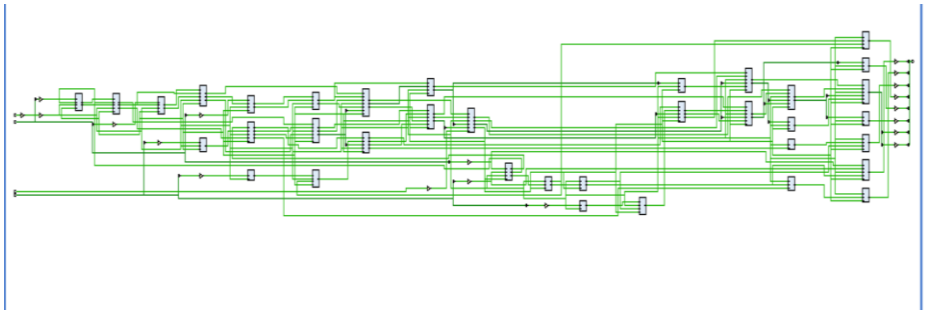


Για παράδειγμα, βλέπουμε ότι για A =14 και B =15 λαμβάνουμε το αποτέλεσμα μετά από 9 κύκλους και αυτό είναι το 210.



Σε περίπτωση υπερχείλισης το P7 βγάζει συγχρονισμένα σωστό αποτέλεσμα

Εύρεση του critical path :



Path 1	∞	2	2	1	delay_P0_C/delay1/q_reg/C	P[0]	4.076	3.276	0.800	∞
Path 2	∞	2	2	1	delay_P1_C/q_reg/C	P[1]	4.076	3.276	0.800	∞
Path 3	∞	2	2	1	delay_P2_B/delay4/q_reg/C	P[2]	4.076	3.276	0.800	∞
Path 4	∞	2	2	1	delay_P3/delay1/q_reg/C	P[3]	4.076	3.276	0.800	∞
Path 5	∞	2	2	1	delay_P4/delay4/q_reg/C	P[4]	4.076	3.276	0.800	∞
Path 6	∞	2	2	1	delay_P5/q_reg/C	P[5]	4.076	3.276	0.800	∞
Path 7	∞	2	2	1	fa_15/out_temp_reg[0]/C	P[6]	4.076	3.276	0.800	∞
Path 8	∞	2	2	1	fa_15/out_temp_reg[1]/C	P[7]	4.076	3.276	0.800	∞

Παρατηρούμε ότι το critical path είναι το ίδιο (**4.076ns**) για την μετάβαση σε καθεμιά από τις εξόδους του πολλαπλασιαστή (P[0] – P[7]). Συγκεκριμένα, για τις εξόδους P[0] ως P[5] η εν λόγω καθυστέρηση ξεκινά από την είσοδο του εκάστοτε DFF, ενώ για τις εξόδους P[6] & P[7] από την είσοδο τελευταίου FA*, μιας και οι εν λόγω έξοδοι βγαίνουν τελευταίες στον pipelined αθροιστή και δεν χρειάζονται περαιτέρω καθυστέρηση με DFFs.