A black background with a black square

Description automatically generated with medium confidence **Εθνικό Μετσόβιο Πολυτεχνείο**

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εικόνα που περιέχει ηλεκτρονικές συσκευές, ηλεκτρονικό εξάρτημα, στοιχείο κυκλώματος, παθητικό στοιχείο κυκλώματος

Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο.

**Μάθημα: Ψηφιακά Συστήματα VLSI**

**Θέμα: 3η Εργαστηριακή Άσκηση – Σχεδίαση Μονάδων Υλικού με την Τεχνική Pipelining**

Εξάμηνο: 8ο

Ομάδα: 6

Συνεργάτες:

* Ακύλας Αντώνιος 03121152
* Κουμπιάς Ιωάννης-Χρυσοβαλάντης 03121053

**Άσκηση 1**

Στο πρώτο κομμάτι του εργαστηρίου σχεδιάσαμε έναν Σύγχρονο Πλήρη Αθροιστή (Full Adder - FA) σε περιγραφή συμπεριφοράς (Behavioral). Η κύρια διαφορά με τον πλήρη αθροιστή που σχεδιάσαμε στο προηγούμενο εργαστήριο είναι η εξαγωγή του αποτελέσματος με τη θετική ακμή του ρολογιού. Συνεπώς προστίθενται registers στην έξοδο για την αποθήκευση του αθροίσματος και του κρατουμένου. Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του κυκλώματος που προκύπτει:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity fa\_sync is

    port(

        a: in std\_logic;

        b: in std\_logic;

        cin: in std\_logic;

        clk: in std\_logic;

        s: out std\_logic;

        cout: out std\_logic

    );

end fa\_sync;

architecture behavioral of fa\_sync is

    signal result: std\_logic\_vector(1 downto 0);

begin

    sync\_add: process(clk)

    begin

        if clk'event and clk='1' then

            result <= ('0' & a) + ('0' & b) + ('0' & cin);

        end if;

    end process;

    s <= result(0);

    cout <= result(1);

end behavioral;

A diagram of a computer circuit

AI-generated content may be incorrect.

Στην συνέχεια δημιουργήσαμε κατάλληλο testbench που παράγει όλα τα διαφορετικά inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και τα αποτελέσματα του simulation:

library ieee;

use ieee.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

entity fa\_sync\_tb is

end fa\_sync\_tb;

architecture fa\_sync\_tb\_arch of fa\_sync\_tb is

    component fa\_sync

        port(

            a: in std\_logic;

            b: in std\_logic;

            cin: in std\_logic;

            clk: in std\_logic;

            s: out std\_logic;

            cout: out std\_logic

        );

    end component;

    constant CLKP : time := 1ns;

    signal in\_s: std\_logic\_vector(2 downto 0);

    signal s\_s,c\_s,clk: std\_logic;

begin

    UUT: fa\_sync port map (in\_s(0),in\_s(1),in\_s(2),clk,s\_s,c\_s);

    clk\_proc: process

    begin

        clk <= '0';

        wait for CLKP/2;

        clk <= '1';

        wait for CLKP/2;

    end process;

    testSequence: process

    begin

        in\_s <= "000";

        for i in 0 to 7 loop

            wait for CLKP;

            in\_s <= in\_s + 1;

        end loop;

        wait;

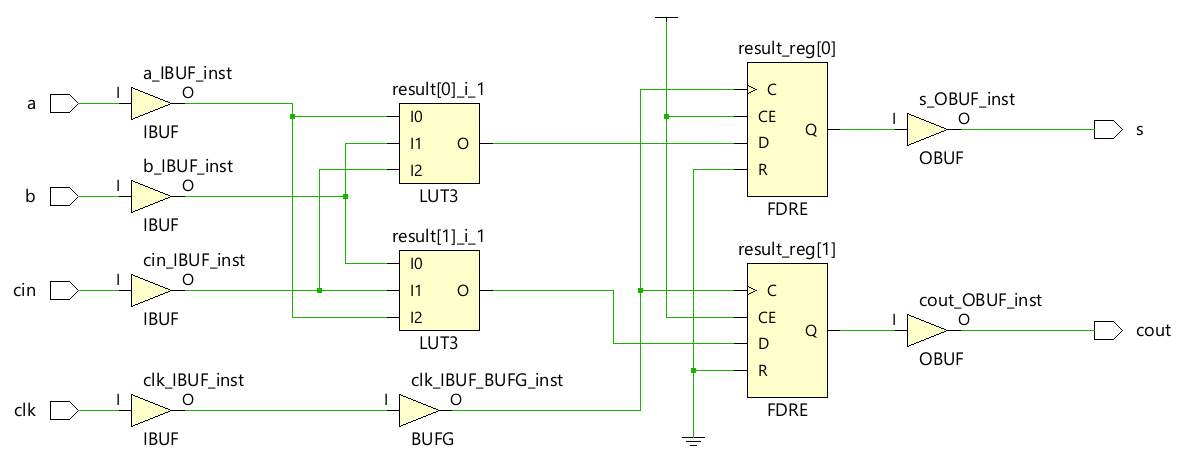
    end process;

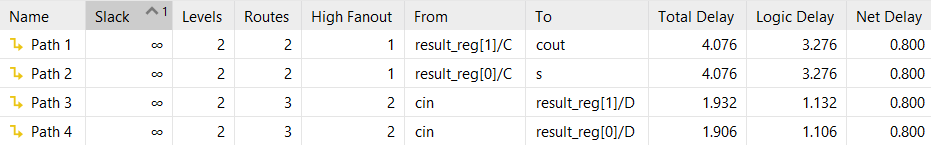
end fa\_sync\_tb\_arch;

A screenshot of a video game

AI-generated content may be incorrect.

Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ’ επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:





Το critical path της σύνθεσης φάινεται πρώτο στα Unconstrained Paths του timing report. Στις μετρήσεις συνυπολογίζονται τόσο η καθυστέρηση λόγω λογικής (Logic Delay) όσο και η καθυστέρηση λόγω του routing των συνδέσεων στο FPGA (Net Delay). Παρατηρούμε πως τα κρίσιμα μονοπάτια είναι από τον καταχωρητή result\_reg[1] προς το output cout και από τον καταχωρητή result\_reg[2] προς το output s. Ακολουθούν τα μονοπάτια cin – result\_reg[1] και cin – result\_reg[0] με μικρότερη καθυστέρηση που οφείλεται στη λογική.

**Άσκηση 2**

Στο δεύτερο κομμάτι του εργαστηρίου σχεδιάσαμε έναν Σύγχρονο Αθροιστή διάδοσης κρατουμένου των 4 bits με χρήση της τεχνικής pipeline. Η διαφορά με τον Αθροιστή της προηγούμενης εργαστηριακής άσκησης είναι πως οι αθροιστές δεν εξάγουν τα αποτελέσματά τους κατευθείαν στην έξοδο, αλλά σε καταχωρητές, οι οποίοι τα διαδίδουν στο pipeline έως ότου να πρέπει να γίνει η εξαγωγή στην έξοδο την κατάλληλη χρονική στιγμή. Έτσι όταν ένας πλήρης αθροιστής ως μονάδα τελείωσει με έναν υπολογισμό και διαδόσει το κρατούμενο στον επόμενο, μπορεί να στείλει το άθροισμα σε καταχωρητή και να ξεκινήσει στον αμέσως επόμενο κύκλο ρολογιού έναν νέο υπολογισμό. Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του κυκλώματος που προκύπτει:

library ieee;

use ieee.std\_logic\_1164.all;

entity rca4\_pipe 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;

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

        cout: out std\_logic

    );

end rca4\_pipe;

architecture rca4\_arch of rca4\_pipe is

    component fa\_sync is

        port(

            a: in std\_logic;

            b: in std\_logic;

            cin: in std\_logic;

            clk: in std\_logic;

            s: out std\_logic;

            cout: out std\_logic

        );

    end component;

    signal r1: std\_logic\_vector(5 downto 0) := (others => '0');

    signal r2: std\_logic\_vector(4 downto 0) := (others => '0');

    signal r3: std\_logic\_vector(3 downto 0) := (others => '0');

    signal r4: std\_logic\_vector(2 downto 0) := (others => '0');

    signal s\_t: std\_logic\_vector(2 downto 0);

    signal c\_t: std\_logic\_vector(2 downto 0);

begin

    pipeline: process(clk)

    begin

        if clk'event and clk='1' then

            r4 <= s\_t(2) & r3(1 downto 0);

            r3 <= r2(4 downto 3) & s\_t(1) & r2(0);

            r2 <= r1(5 downto 2) & s\_t(0);

            r1 <= b(3) & a(3) & b(2) & a(2) & b(1) & a(1);

        end if;

    end process;

    fa1: fa\_sync port map(a(0),b(0),cin,clk,s\_t(0),c\_t(0));

    fa2: fa\_sync port map(r1(0),r1(1),c\_t(0),clk,s\_t(1),c\_t(1));

    fa3: fa\_sync port map(r2(1),r2(2),c\_t(1),clk,s\_t(2),c\_t(2));

    fa4: fa\_sync port map(r3(2),r3(3),c\_t(2),clk,s(3),cout);

    s(2 downto 0) <= r4(2 downto 0);

end rca4\_arch;

A diagram of a computer

AI-generated content may be incorrect.

Στην συνέχεια δημιουργήσαμε κατάλληλο testbench που παράγει όλα τα διαφορετικά inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και κάποια αντιπροσωπευτικά αποτελέσματα του simulation:

library ieee;

use ieee.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

entity rca4\_pipe\_tb is

end rca4\_pipe\_tb;

architecture rca4\_pipe\_tb\_arch of rca4\_pipe\_tb is

    component rca4\_pipe 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;

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

            cout: out std\_logic

        );

    end component;

    constant CLKP : time := 1ns;

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

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

    signal cin\_s: std\_logic;

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

    signal cout\_s: std\_logic;

    signal clk: std\_logic;

begin

    clk\_proc: process

    begin

        clk <= '0';

        wait for CLKP/2;

        clk <= '1';

        wait for CLKP/2;

    end process;

    UUT: rca4\_pipe port map (a\_s,b\_s,cin\_s,clk,s\_s,cout\_s);

    testSequence: process

        begin

            a\_s <= "0000";

            b\_s <= "0000";

            cin\_s <= '0';

            for i in 0 to 15 loop

                for j in 0 to 15 loop

                    wait for CLKP;

                    a\_s <= a\_s + 1;

                end loop;

                b\_s <= b\_s + 1;

            end loop;

            cin\_s <= '1';

            for i in 0 to 15 loop

                for j in 0 to 15 loop

                    wait for CLKP;

                    a\_s <= a\_s + 1;

                end loop;

                b\_s <= b\_s + 1;

            end loop;

            wait;

    end process;

end rca4\_pipe\_tb\_arch;

A screenshot of a video game

AI-generated content may be incorrect.

A black and green screen with white squares

AI-generated content may be incorrect.

A computer screen with green and black squares

AI-generated content may be incorrect.

A screen shot of a video game

AI-generated content may be incorrect.

Αξίζει να παρατηρήσουμε πως μεσολαβούν 3 κύκλοι ρολογιού μέχρι να ξεκινήσει να εξάγει έγκυρα αποτελέσματα ο αθροιστής. Αυτό οφείλεται στο βάθος του pipeline το οποίο παράγει έγκυρα αποτελέσματα σε κάθε κύκλο ρολογιού αφού έχει γίνει και ο υπολογισμός του 4ου και τελευταίου bit του πρώτου αποτελέσματος.

Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ’ επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:

A diagram of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A diagram of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Παρατηρώ πως η χρήση της τεχνικής του pipeline για την κατασκευή του αθροιστή 4 bits αυξάνει σημαντικά την πολυπλοκότητα του κυκλώματος, οδηγώντας σε περεταίρω κατανάλωση πόρων κυρίως λόγω των registers που επιτυγχάνουν τον συγχρονισμό των δεδομένων. Ωστόσο όπως φαίνεται και από το timing summary με τα critical paths, το path στον pipeline αθροιστή είναι σημαντικά μικρότερο από αυτό στον απλό, αφού στην δεύτερη περίπτωση περιοριζόμαστε από το dataflow. Συνεπώς με την τεχνική του pipeline, δημιουργήσαμε ένα σαφώς μεγαλύτερο κύκλωμα με καθυστέρηση κατά την εκκίνησή του, που όμως πετυχαίνει σημαντικά καλύτερο throughput.

**Άσκηση 3**

Στο τρίτο και τελευταίο κομμάτι του εργαστηρίου σχεδιάσαμε έναν Συστολικό Πολλαπλασιαστή διάδοσης κρατουμένων των 4 bits κάνοντας χρήση των σύγχρονων Πλήρων Αθροιστών της πρώτης άσκησης. Για την δημιουργία ενός απλού συνεχούς pipeline θα αρκούσε η εισαγωγή καθυστερήσεων μεταξύ των κυττάρων ενός κανονικού πολλαπλασιαστή 4 bits. Ωστόσο αν και προκύπτει το ίδιο αποτέλεσμα στο συγκεκριμένο κύκλωμα, επιλέξαμε να παραμείνουμε τυπικοί στον ορισμό του συστολικού pipeline και πρώτα σχεδιάσαμε ένα συστολικό κύτταρο, με την επανάληψη του οποίου μπορούμε να κατασκευάσουμε το μεγαλύτερο μέρος της λειτουργικότητας του πολλαπλασιαστή. Παρουσιάζουμε τον κώδικα VHDL καθώς και το RTL Schematic του συστολικού κυττάρου:

library ieee;

use ieee.std\_logic\_1164.all;

entity mul\_cell is

    port(

        ain: in std\_logic;

        bin: in std\_logic;

        cin: in std\_logic;

        sin: in std\_logic;

        clk: in std\_logic;

        aout: out std\_logic;

        bout: out std\_logic;

        cout: out std\_logic;

        sout: out std\_logic

    );

end mul\_cell;

architecture mul\_cell\_arch of mul\_cell is

    component fa\_sync is

        port(

        a: in std\_logic;

        b: in std\_logic;

        cin: in std\_logic;

        clk: in std\_logic;

        s: out std\_logic;

        cout: out std\_logic

    );

    end component;

    signal a\_inter,t: std\_logic;

begin

    sync\_mul: process(clk)

    begin

        if clk'event and clk='1' then

            aout <= a\_inter;

            a\_inter <= ain;

            bout <= bin;

        end if;

    end process;

    t <= ain and bin;

    fa: fa\_sync port map(

        a => sin,

        b => t,

        cin => cin,

        clk => clk,

        s => sout,

        cout => cout

    );

end mul\_cell\_arch;

A diagram of a computer

AI-generated content may be incorrect.

Στην συνέχεια, αξιοποιώντας 16 τέτοια κύτταρα καθώς και επιπλέον καταχωρητές για τον σωστό συγχρονισμό του pipeline ως προς την διάδοση των inputs a και b, των εξωτερικών κρατουμένων cout καθώς και των γινομένων σχεδιάσαμε ολόκληρο τον συστολικο πολλαπλασιαστή. Ακολουθεί ο κώδικας VHDL καθώς και το RTL Schematic του κυκλώματος:

library ieee;

use ieee.std\_logic\_1164.all;

entity mul\_syst is

    port(

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

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

        clk: in std\_logic;

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

    );

end mul\_syst;

architecture mul\_syst\_arch of mul\_syst is

    component mul\_cell is

        port(

        ain: in std\_logic;

        bin: in std\_logic;

        cin: in std\_logic;

        sin: in std\_logic;

        clk: in std\_logic;

        aout: out std\_logic;

        bout: out std\_logic;

        cout: out std\_logic;

        sout: out std\_logic

    );

    end component;

    -- signals

    signal ai, bi, ci, si, ao, bo, co, so: std\_logic\_vector(15 downto 0);

    -- horizontal registers (by levels of deep)

    signal r\_b1: std\_logic\_vector(1 downto 0);

    signal r\_b2: std\_logic\_vector(3 downto 0);

    signal r\_b3: std\_logic\_vector(5 downto 0);

    -- diagonal registers (by levels of deep)

    signal r\_d1: std\_logic\_vector(2 downto 0);

    signal r\_d2: std\_logic\_vector(1 downto 0);

    signal r\_d3: std\_logic;

    -- outer carry register (gtsm)

    signal rc: std\_logic\_vector(2 downto 0);

    -- vertical register array (by bit position)

    type reg\_array is array (0 to 5) of std\_logic\_vector(8 downto 0);

    signal r: reg\_array;

begin

    --pipeline gtsm (c outer)

    pipeline\_outer: process(clk)

    begin

        if clk'event and clk='1' then

            rc <= co(11) & co(7) & co(3);

        end if;

    end process;

    --pipeline diagonal (a)

    pipeline\_diag: process(clk)

    begin

        if clk'event and clk='1' then

            r\_d3 <= r\_d2(1);

            r\_d2 <= r\_d1(2 downto 1);

            r\_d1 <= a(3 downto 1);

        end if;

    end process;

    --pipeline horizontal (b)

    pipeline\_hor: process(clk)

    begin

        if clk'event and clk='1' then

            for i in 5 downto 1 loop

                r\_b3(i) <= r\_b3(i-1);

            end loop;

            r\_b3(0) <= b(3);

            for i in 3 downto 1 loop

                r\_b2(i) <= r\_b2(i-1);

            end loop;

            r\_b2(0) <= b(2);

            r\_b1(1) <= r\_b1(0);

            r\_b1(0) <= b(1);

        end if;

    end process;

    -- pipeline vertical

    pipeline\_vert: process(clk)

    begin

        if clk'event and clk='1' then

            for ii in 0 to 5 loop

                for jj in 8 downto 1 loop

                    r(ii)(jj) <= r(ii)(jj-1);

                end loop;

                if ii < 3 then

                    r(ii)(0) <= so(4\*ii);

                else

                    r(ii)(0) <= so(9+ii);

                end if;

            end loop;

        end if;

    end process;

    -- cells init

    ginit: for i in 0 to 15 generate

        gcell: mul\_cell port map(ai(i), bi(i), ci(i), si(i), clk, ao(i), bo(i), co(i), so(i));

    end generate;

    -- inter wires

    ga: for i in 4 to 15 generate

       ai(i) <= ao(i-4);

    end generate;

    ci(0) <= '0';

    ci(4) <= '0';

    ci(8) <= '0';

    ci(12) <= '0';

    gbci: for i in 0 to 3 generate

        gbcj: for j in 1 to 3 generate

            bi(4\*i+j) <= bo(4\*i+j-1);

            ci(4\*i+j) <= co(4\*i+j-1);

        end generate;

    end generate;

    si(3 downto 0) <= "0000";

    gs: for i in 1 to 3 generate

        si(4\*i) <= so(4\*(i-1)+1);

        si(4\*i+1) <= so(4\*(i-1)+2);

        si(4\*i+2) <= so(4\*(i-1)+3);

        si(4\*i+3) <= rc(i-1);

    end generate;

    -- input

    ai(0) <= a(0);

    ai(1) <= r\_d1(0);

    ai(2) <= r\_d2(0);

    ai(3) <= r\_d3;

    bi(0) <= b(0);

    bi(4) <= r\_b1(1);

    bi(8) <= r\_b2(3);

    bi(12) <= r\_b3(5);

    -- result

    p <= co(15) & so(15) & r(5)(0) & r(4)(1) & r(3)(2) & r(2)(4) & r(1)(6) & r(0)(8);

end mul\_syst\_arch;

A diagram of a computer network

AI-generated content may be incorrect.

Στην συνέχεια δημιουργήσαμε κατάλληλο testbench που παράγει όλα τα διαφορετικά inputs για τον έλεγχο της ορθής λειτουργίας του κυκλώματος. Ακολουθούν ο κώδικας VHDL του testbench καθώς και κάποια αντιπροσωπευτικά αποτελέσματα του simulation:

library ieee;

use ieee.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

entity mul\_syst\_tb is

end mul\_syst\_tb;

architecture mul\_syst\_tb\_arch of mul\_syst\_tb is

    component mul\_syst is

        port(

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

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

            clk: in std\_logic;

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

        );

    end component;

    constant CLKP : time := 1ns;

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

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

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

    signal clk: std\_logic;

begin

    clk\_proc: process

    begin

        clk <= '0';

        wait for CLKP/2;

        clk <= '1';

        wait for CLKP/2;

    end process;

    UUT: mul\_syst port map (a\_s,b\_s,clk,p\_s);

    testSequence: process

        begin

            a\_s <= "0000";

            b\_s <= "0000";

            for i in 0 to 15 loop

                for j in 0 to 15 loop

                    wait for CLKP;

                    a\_s <= a\_s + 1;

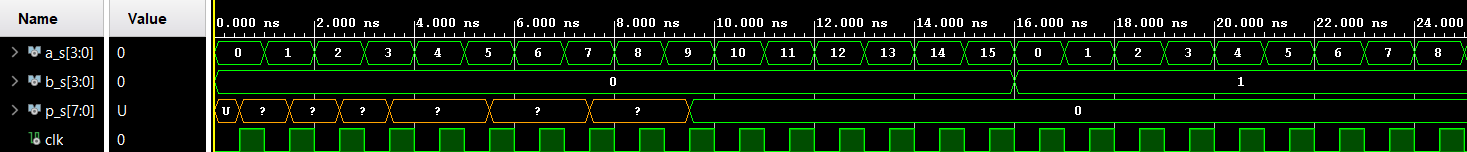
                end loop;

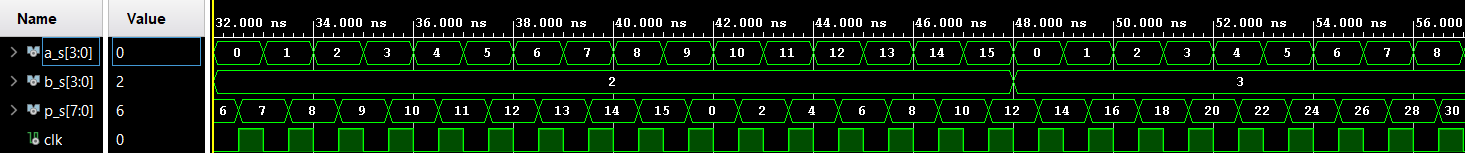
                b\_s <= b\_s + 1;

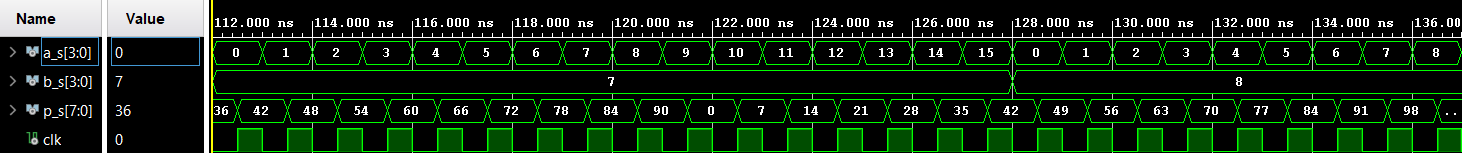
            end loop;

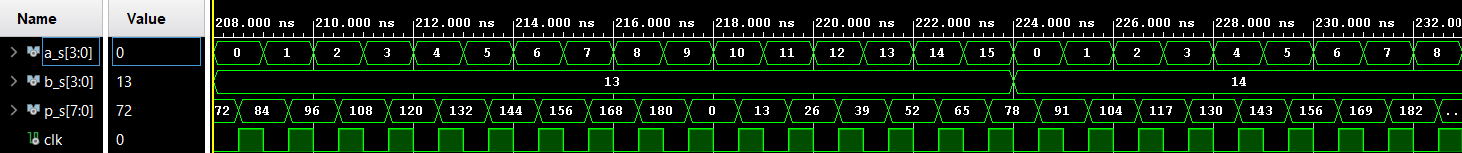
    end process;

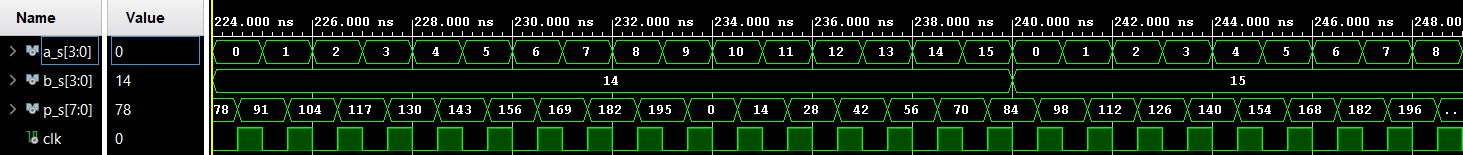
end mul\_syst\_tb\_arch;





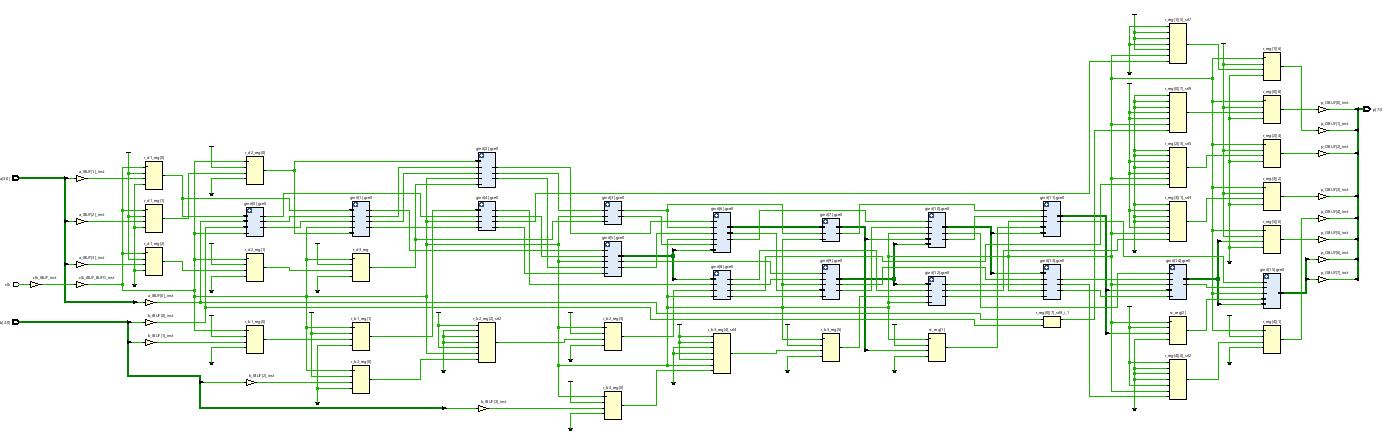






Αξίζει να παρατηρήσουμε πως μεσολαβούν 9 κύκλοι ρολογιού μέχρι να ξεκινήσει να εξάγει έγκυρα αποτελέσματα ο πολλαπλασιαστής. Αυτό οφείλεται στο βάθος του pipeline το οποίο έχει μέγιστο βάθος 10 για να γίνει σωστός συγχρονισμός των δεδομένων μέχρι την παραγωγή του κάθε αποτελέσματος.

Έχοντας βεβαιωθεί πως το κύκλωμα λειτουργεί όπως είναι αναμενόμενο, προχωράμε στη διαδικασία σύνθεσης με στόχο τον υπολογισμό του critical path και κατ’ επέκταση της μέγιστης χρονικής καθυστέρησης με την βοήθεια του Vivado. Παραθέτουμε το schematic της σύνθεσης αξιοποιώντας πλέον πόρους του FPGA και τις μετρήσεις των χρονικών καθυστερήσεων ανά μονοπάτι:



A screenshot of a computer

AI-generated content may be incorrect.

Παρατηρώ στο timing summary πως η χρονική διάρκεια των critical paths είναι εντυπωσιακά μικρή χάρη στην

παρεμβολή registers μεταξύ των δομικών μονάδων του κυκλώματος και του κατάλληλου συγχρονισμού. Στην

περίπτωση που υλοποιούσαμε το κύκλωμα χωρίς την τεχνική του pipeline θα είχαμε μικρότερη κατανάλωση πόρων

κυρίως σε registers και μικρότερη πολυπλοκότητα (και ίσως μικρότερο net delay), ωστόσο θα υπήρχε μεγάλη

καθυστέρηση εξαιτίας του dataflow και της μειωμένης συχνότητας ρολογιού που θα απαιτούσε. Αντιθέτως στην

προκειμένη περίπτωση η παρεμβολή registers και η χρήση συστολικών κυττάρων αυξάνει σημαντικά το τελικό

throughput του κυκλώματος.