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: 4η Εργαστηριακή Άσκηση – Υλοποίηση FIR Φίλτρου**

Εξάμηνο: 8ο

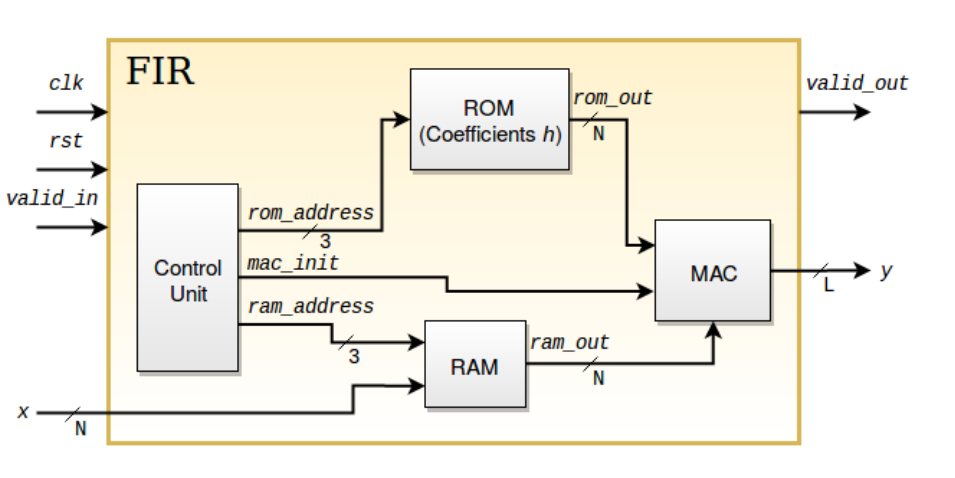
Ομάδα: 6

Συνεργάτες:

* Ακύλας Αντώνιος 03121152
* Κουμπιάς Ιωάννης-Χρυσοβαλάντης 03121053

**Εισαγωγή**

Σκοπός αυτής της εργαστηριακής άσκησης είναι η υλοποίηση ενός διακριτού φίλτρου FIR 8 τιμών. Παρουσιάζουμε τόσο τη μαθηματική σχέση που διέπει το σύστημα όσο και ένα ποιοτικό σχέδιο της αρχιτεκτονικής του συτήματός μας:



Συνεπώς το ζητούμενο της άσκησης είναι η υλοποίηση καθενός από τα 4 modules που φαίνονται στο σχεδιάγραμμα καθώς και ο συγχρονισμός τους, ώστε να έχουμε ένα λειτουργικό φίλτρο.

**Multiplier Accumulator Unit (MAC)**

Η μονάδα αυτή δέχεται ως είσοδο δύο 8-bit αριθμούς, τους πολλαπλασιάζει και τους προσθέτει σε έναν εσωτερικό accumulator. Για τον έλεγχο του της μονάδας χρησιμοποιείται και ένα σήμα αρχικοποίησης mac\_init, το οποίο όταν είναι ‘1’ μηδενίζει την τιμή που είναι αποθηκευμένη στον accumulator, ξεκινώντας έτσι την παραγωγή ενός νέου αποτελέσματος. Η περιγραφή του module σε VHDL καθώς και το RTL schematic φαίνονται παρακάτω :

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity mac is

port(

init: in std\_logic;

b: in std\_logic\_vector(7 downto 0);

c: in std\_logic\_vector(7 downto 0);

clk: in std\_logic;

y: out std\_logic\_vector(18 downto 0) -- 2X + 3 bits

);

end mac;

architecture behavioral of mac is

signal acc: std\_logic\_vector(18 downto 0) := (others => '0'); --accumulator

begin

calc: process(clk)

begin

if clk'event and clk = '1' then

if init = '1' then

acc <= (others => '0');

acc(15 downto 0) <= b \* c;

else

acc <= acc + (b \* c);

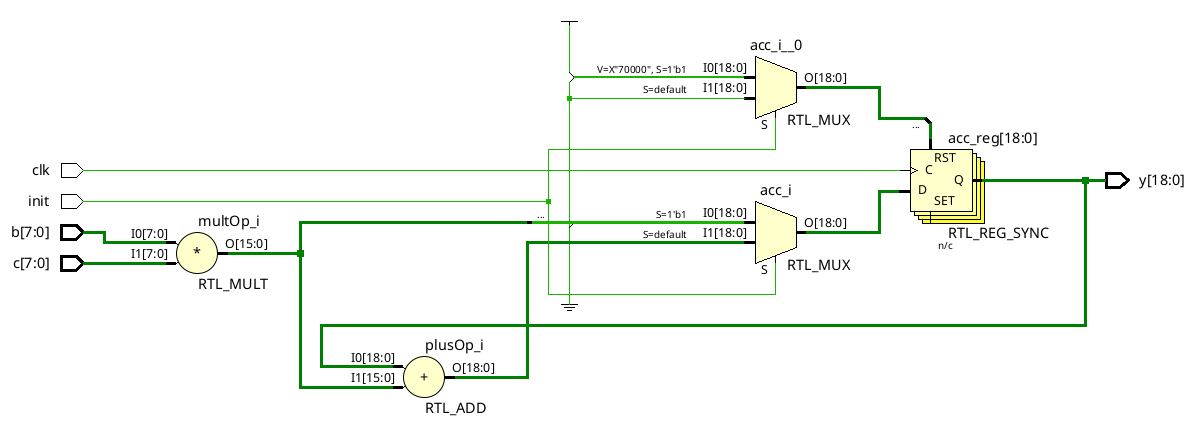
end if;

end if;

end process;

y <= acc;

end behavioral;



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.NUMERIC\_STD.ALL;

entity mac\_tb is

generic (

data\_width : integer :=8 --- width of data (bits)

);

end mac\_tb;

architecture test\_bench of mac\_tb is

component mac is

Port (

init: in std\_logic;

b: in std\_logic\_vector(7 downto 0);

c: in std\_logic\_vector(7 downto 0);

clk: in std\_logic;

y: out std\_logic\_vector(18 downto 0) -- 2X + 3 bits

);

end component;

signal clk : std\_logic;

signal init: std\_logic;

signal b : std\_logic\_vector(data\_width-1 downto 0);

signal c : std\_logic\_vector(data\_width-1 downto 0);

signal y : std\_logic\_vector(2\*data\_width+3-1 downto 0);

constant CLOCK\_PERIOD : time := 10 ns;

begin

MAC\_INST: mac

port map(init, b, c, clk, y);

STIMULUS: process

begin

init <= '0';

b <= std\_logic\_vector(to\_unsigned(2,8));

for i in 0 to 8 loop

c <= std\_logic\_vector(to\_unsigned(i,8));

wait for (1\*CLOCK\_PERIOD);

end loop;

init <= '1';

wait for CLOCK\_PERIOD;

init <= '0';

for i in 0 to 8 loop

c <= std\_logic\_vector(to\_unsigned(i,8));

b <= std\_logic\_vector(to\_unsigned(i,8));

wait for (1\*CLOCK\_PERIOD);

end loop;

wait;

end process;

clk\_proc: process

begin

clk <= '0';

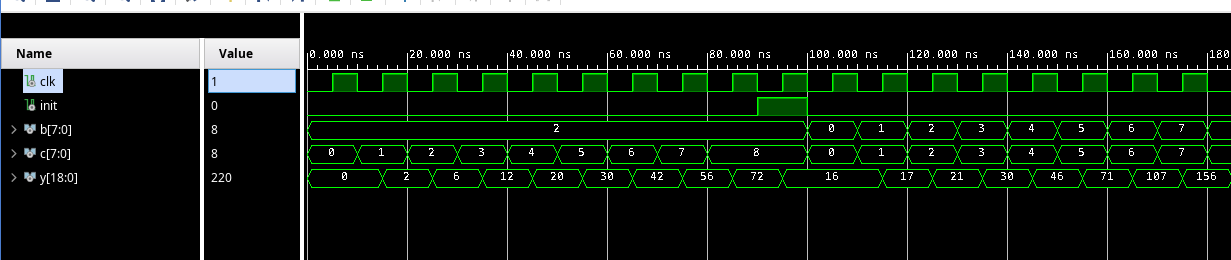
wait for CLOCK\_PERIOD/2;

clk <= '1';

wait for CLOCK\_PERIOD/2;

end process;

end test\_bench;



**Read Only Memory (ROM)**

Η λειτουργία του φίλτρου βασίζεται στη συνέλιξη των n τελευταίων τιμών (8 στην περίπτωσή μας) του σήματος εισόδου x με τους σταθερούς συντελεστές h. Για την αποθήκευση των παραπάνω συντελεστών υλοποιούμε μία μονάδα μνήμης ROM, η οποία δέχεται ως είσοδο μία διεύθυνση (3 bit) και δίνει στην έξοδό της την αντίστοιχη αποθηκευμένη σταθερά. Η περιγραφή του module σε VHDL καθώς και το RTL schematic φαίνονται παρακάτω :

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

entity mlab\_rom is

generic (

coeff\_width : integer :=8 --- width of coefficients (bits)

);

Port (

clk : in STD\_LOGIC;

en : in STD\_LOGIC; --- operation enable

addr : in STD\_LOGIC\_VECTOR (2 downto 0); -- memory address

rom\_out : out STD\_LOGIC\_VECTOR (coeff\_width-1 downto 0)); -- output data

end mlab\_rom;

architecture Behavioral of mlab\_rom is

type rom\_type is array (7 downto 0) of std\_logic\_vector (coeff\_width-1 downto 0); -- initialization of rom with user data

signal rom : rom\_type:= ("00001000", "00000111", "00000110", "00000101", "00000100", "00000011", "00000010", "00000001");

signal rdata : std\_logic\_vector(coeff\_width-1 downto 0) := (others => '0');

begin

rdata <= rom(conv\_integer(addr));

process (clk)

beginδδδ if (clk'event and clk = '1') then

if (en = '1') then

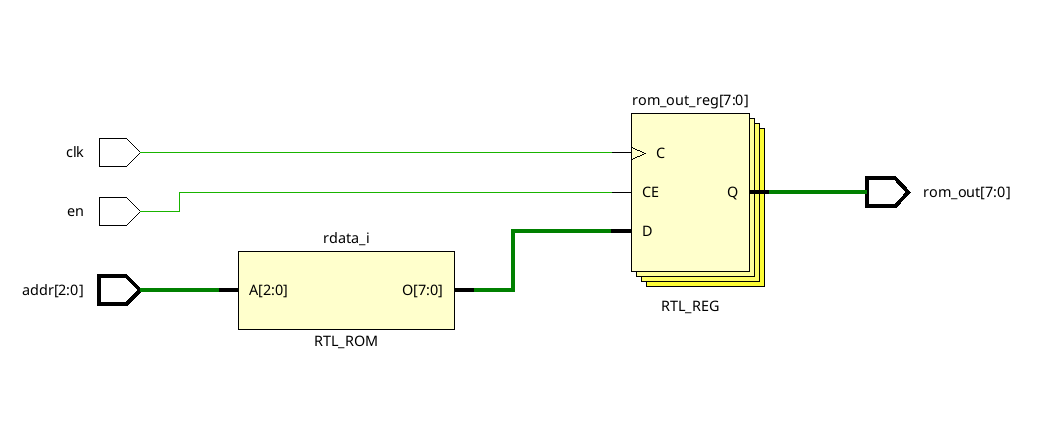
rom\_out <= rdata;

end if;

end if;

end process;

end Behavioral;



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.NUMERIC\_STD.ALL;

use ieee.std\_logic\_unsigned.all;

entity mlab\_rom\_tb is

generic (

coeff\_width : integer :=8

);

end mlab\_rom\_tb;

architecture test\_bench of mlab\_rom\_tb is

component mlab\_rom is

Port (

clk : in STD\_LOGIC;

en : in STD\_LOGIC; --- operation enable

addr : in STD\_LOGIC\_VECTOR (2 downto 0); -- memory address

rom\_out : out STD\_LOGIC\_VECTOR (coeff\_width-1 downto 0)); -- output data

end component;

signal clk : std\_logic;

signal en : std\_logic;

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

signal rom\_out : std\_logic\_vector(coeff\_width-1 downto 0);

constant CLOCK\_PERIOD : time := 10 ns;

begin

rom: mlab\_rom

port map(clk,en,addr,rom\_out);

STIMULUS: process

begin

en <= '1';

addr <= "000";

for i in 0 to 10 loop

addr <= std\_logic\_vector(to\_unsigned(i,3));

wait for (1 \* CLOCK\_PERIOD);

end loop;

wait;

end process;

clk\_proc: process

begin

clk <= '0';

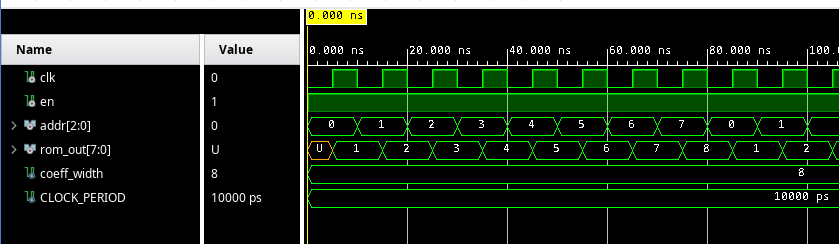
wait for CLOCK\_PERIOD/2;

clk <= '1';

wait for CLOCK\_PERIOD/2;

end process;

end test\_bench;

****

Όπως βλέπουμε παραπάνω, η ROM επιστρέφει τις τιμές που είναι αποθηκευμένες στην επιθυμητή διεύθυνση σε κάθε θετική ακμή του ρολογιού.

**Random Access Memory (RAM)**

Όπως αναφέραμε παραπάνω, για τον υπολογισμό της εξόδου του φίλτρου χρειαζόμαστε τις τελευταίες 8 τιμές του σήματος, είναι λογικό λοιπόν να χρησιμοποιήσουμε μία μονάδα μνήμης για την αποθήκευσή τους. Κατ’ αναλογία με τη ROM, η RAM δέχεται μία διεύθυνση ως είσοδο και δίνει στην έξοδό της την αντίστοιχη αποθηκευμένη τιμή. Ωστόσο, για την RAM υπάρχει και η δυνατότητα εγγραφής δεδομένων. Πιο συγκεκριμένα, ένα σήμα we ενεργοποιεί τη λειτουργία εγγραφής, επιτρέποντας την εγγραφή των τιμών εισόδου στην επιλεγμένη διεύθυνση της μνήμης. Ωστόσο, για τις ανάγκες του φίλτρου είναι απαραίτητη επιπλέον λειτουργικότητα του κυκλώματος.

Αρχικά, για να αποφύγουμε επιπρόσθετη καθυστέρηση, χρησιμοποιούμε write-first RAM με την οποία έχουμε πρόσβαση στα περιεχόμενα της μνήμης στον ίδιο κύκλο ρολογιού που έγινε η εγγραφή τους.

Ακόμα, υλοποιούμε ένα σύστημα ολίσθησης στη μνήμη, με σκοπό να κρατάμε κάθε στιγμή μόνο τις 8 τελευταίες τιμές και να απορρίπτουμε την παλαιότερη όταν κάποια νέα είναι διαθέσιμη. Τέλος, για τον συνολικό έλεγχο της αρχικοποίησης του κυκλώματος προσθέτουμε ένα ασύγχρονο σήμα αρχικοποίησης (reset) το οποίο όταν τεθεί μηδενίζει τα περιεχόμενα της μνήμης. Η περιγραφή του module σε VHDL καθώς και το RTL schematic φαίνονται παρακάτω:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity mlab\_ram is

generic (

data\_width : integer :=8 --- width of data (bits)

);

port (

rst : in std\_logic;

clk : in std\_logic;

we : in std\_logic; --- memory write enable

en : in std\_logic; --- operation enable

addr : in std\_logic\_vector(2 downto 0); -- memory address

di : in std\_logic\_vector(data\_width-1 downto 0); -- input data

do : out std\_logic\_vector(data\_width-1 downto 0)); -- output data

end mlab\_ram;

architecture Behavioral of mlab\_ram is

type ram\_type is array (7 downto 0) of std\_logic\_vector (data\_width-1 downto 0);

signal RAM : ram\_type := (others => (others => '0'));

begin

process (clk,rst)

begin

if rst = '1' then

RAM <= (others => (others => '0')); -- reset RAM to 0

do <= (others => '0');

else

if clk'event and clk = '1' then

if en = '1' then

if we = '1' then -- write operation

RAM(7 downto 1) <= RAM(6 downto 0);

RAM(0) <= di; -- no need for address it works like a stack

do <= di;

else -- read operation

do <= RAM(conv\_integer(addr)); -- here the address is necessary for MAC calculations

end if;

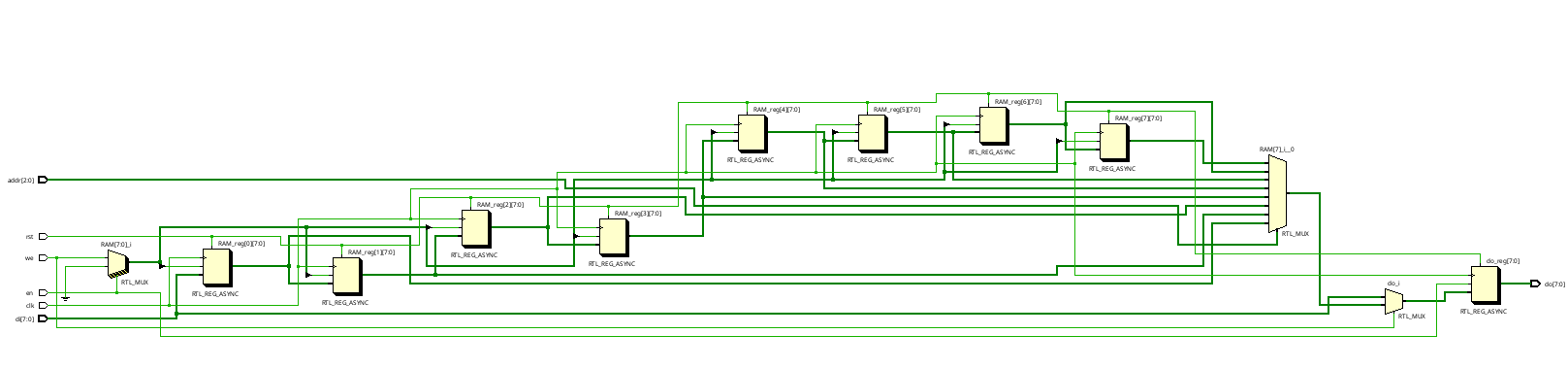
end if;

end if;

end if;

end process;

end Behavioral;



Παρατηρούμε ότι η υλοποίηση του κυκλώματος ακολουθεί τη λογική ενός shift register με έναν πολυπλέκτη για την επιλογή δεδομένων εξόδου. 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.NUMERIC\_STD.ALL;

use ieee.std\_logic\_unsigned.all;

entity mlab\_ram\_tb is

generic (

data\_width : integer :=8

);

end mlab\_ram\_tb;

architecture test\_bench of mlab\_ram\_tb is

component mlab\_ram is

port (

rst : in std\_logic;

clk : in std\_logic;

we : in std\_logic; --- memory write enable

en : in std\_logic; --- operation enable

addr : in std\_logic\_vector(2 downto 0); -- memory address

di : in std\_logic\_vector(data\_width-1 downto 0); -- input data

do : out std\_logic\_vector(data\_width-1 downto 0)); -- output data

end component;

signal clk : std\_logic;

signal rst : std\_logic;

signal we : std\_logic;

signal en : std\_logic;

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

signal di : std\_logic\_vector(data\_width-1 downto 0);

signal do : std\_logic\_vector(data\_width-1 downto 0);

constant CLOCK\_PERIOD : time := 10 ns;

begin

RAM: mlab\_ram

port map(rst,clk,we,en,addr,di,do);

STIMULUS: process

begin

addr <= "000";

rst <= '0';

en <= '1';

we <= '1';

for i in 1 to 10 loop

di <= std\_logic\_vector(to\_unsigned(10\*i,8));

wait for (1 \* CLOCK\_PERIOD);

end loop;

we <= '0';

for i in 0 to 7 loop

addr <= std\_logic\_vector(to\_unsigned(i,3));

wait for (1 \* CLOCK\_PERIOD);

end loop;

rst <= '1';

wait;

end process;

GEN\_CLK : process

begin

clk <= '0';

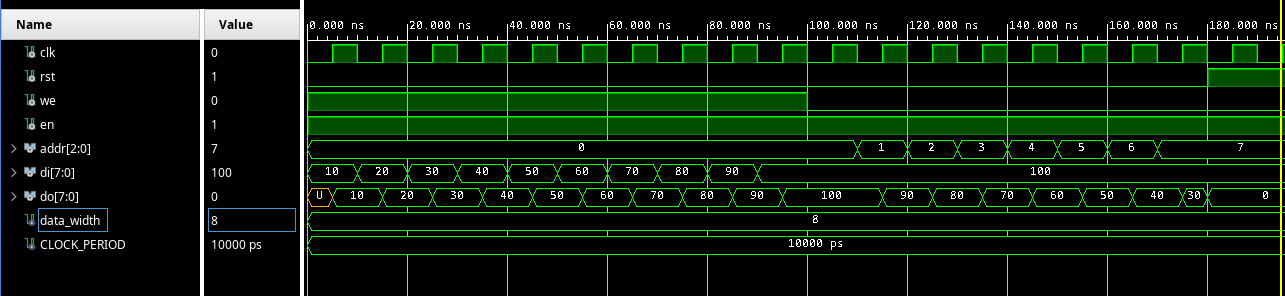
wait for (CLOCK\_PERIOD / 2);

clk <= '1';

wait for (CLOCK\_PERIOD / 2);

end process;

end test\_bench;



Όπως βλέπουμε και παραπάνω όταν we = ‘1’ τα νέα δεδομένα από τη di εγγράφονται στη μνήμη, ενώ όταν we = ’0’ η λειτουργία της RAM ταυτίζεται με αυτή της ROM.

**Control Unit (CU)**

Το τελευταίο module που χρειαζόμαστε είναι μία μονάδα που παράγει όλα τα σήματα για τον έλεγχο των τριών παραπάνω module. Πιο συγκεκριμένα το cu υλοποιεί έναν εσωτερικό μετρητή τριών bit η τιμή του οποίου λειτουργεί ως διεύθυνση για τις μονάδες μνήμης, ενώ αποστέλλει ταυτόχρονα και τα απαραίτητα σήματα ελέγχου των υπόλοιπων μονάδων του κυκλώματος. Συγκεκριμένα, κάθε φορά που γίνεται λήψη μίας νέας και έγκυρης εισόδου (valid\_in = 1), το cu θέτει τα valid\_out (εγκυρότητα εξόδου – αποτελέσματος στον accumulator μετά από 8 κύκλους υπολογισμών), mac\_init (μηδενισμός accumulator για την έναρξη νέου υπολογισμού) και we (σήμα εγγραφής στην RAM). Με τον τρόπο αυτό, ο πλήρης έλεγχος του κυκλώματος γίνεται μέσω του σήματος valid\_in, ώστε το κύκλωμα να ανταπεξέρχεται σε τυχόν καθυστερήσεις των εισόδων, αποφεύγοντας να γεμίζει την μνήμη με ψευδείς εισόδους ή να θεωρεί ενδιάμεσες εξόδους από τον accumulator του mac ως έγκυρες. Η περιγραφή του module σε VHDL καθώς και το RTL schematic φαίνονται παρακάτω:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity control\_unit is

port(

clk: in std\_logic;

rst: in std\_logic;

valid\_in: in std\_logic;

rom\_addr: out std\_logic\_vector(2 downto 0);

ram\_addr: out std\_logic\_vector(2 downto 0);

mac\_init: out std\_logic;

we: out std\_logic;

valid\_out: out std\_logic

);

end control\_unit;

architecture behavioral of control\_unit is

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

begin

addr\_cnt: process(clk,rst)

begin

if rst = '1' then

count <= "000";

else

if clk'event and clk = '1' then

if valid\_in = '1' then

count <= "000";

valid\_out <= '1';

mac\_init <= '1';

we <= '1';

else

count <= count + 1;

valid\_out <= '0';

mac\_init <= '0';

we <= '0';

end if;

end if;

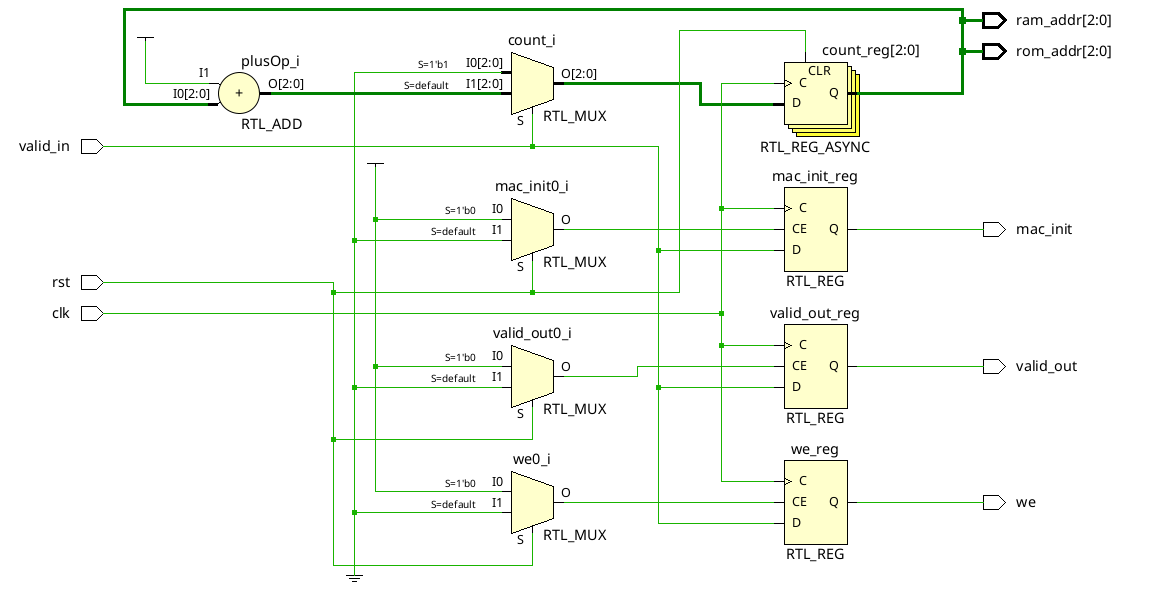
end if;

end process;

rom\_addr <= count;

ram\_addr <= count;

end behavioral;



Στην συνέχεια δημιουργήσαμε κατάλληλο testbench 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.NUMERIC\_STD.ALL;

use ieee.std\_logic\_unsigned.all;

entity control\_unit\_tb is

generic (

data\_width : integer :=8

);

end control\_unit\_tb;

architecture test\_bench of control\_unit\_tb is

component control\_unit is

port(

clk: in std\_logic;

rst: in std\_logic;

valid\_in: in std\_logic;

rom\_addr: out std\_logic\_vector(2 downto 0);

ram\_addr: out std\_logic\_vector(2 downto 0);

mac\_init: out std\_logic;

we: out std\_logic;

valid\_out: out std\_logic

);

end component;

signal clk : std\_logic;

signal rst : std\_logic;

signal valid\_in : std\_logic := '0';

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

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

signal mac\_init : std\_logic;

signal we : std\_logic;

signal valid\_out : std\_logic;

constant CLOCK\_PERIOD : time := 10 ns;

begin

CU: control\_unit

port map(clk,rst,valid\_in,rom\_address,ram\_address,mac\_init,we,valid\_out);

STIMULUS: process

begin

rst <= '1';

wait for CLOCK\_PERIOD;

rst <= '0';

wait for (10\*CLOCK\_PERIOD);

rst <= '1';

wait for CLOCK\_PERIOD;

rst <= '0';

wait for CLOCK\_PERIOD;

valid\_in <= '1';

wait;

end process;

GEN\_CLK : process

begin

clk <= '0';

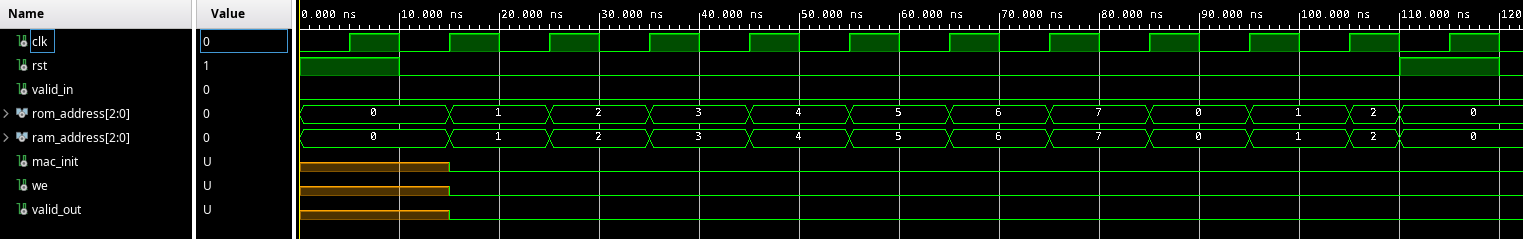
wait for (CLOCK\_PERIOD / 2);

clk <= '1';

wait for (CLOCK\_PERIOD / 2);

end process;

end test\_bench;



Βλέπουμε ότι το κύκλωμα λειτουργεί κανονικά ως μετρητής του οποίο η τιμή μηδενίζεται όταν τεθεί το σήμα rst ή valid\_in. Το πρώτο σήμα αναλαμβάνει την ασύχρονη αρχικοποίηση του κυκλώματος ενώ το δεύτερο σηματοδοτεί την εγκυρότητα μια εισόδους και την επακόλουθη εκκίνηση μίας νέας μέτρησης.

**Υλοποίηση FIR**

Εφόσον έχουμε υλοποιήσει όλα τα απαραίτητα components μπορούμε να φτιάξουμε το τελικό φίλτρο. Πέρα από την ενσωμάτωση όλων των επί μέρους μονάδων πρέπει να εξασφαλίσουμε και τον συγχρονισμό τους.

Για τον λόγο αυτό προσθέτουμε στο κύκλωμα:

* Έναν καταχωρητή μεταξύ του σήματος εισόδου x και της εισόδου της RAM έτσι ώστε τα εισερχόμενα δεδομένα να φθάνουν ταυτόχρονα με το σήμα valid\_in το οποίο μεταδίδεται μέσω του control unit με έναν κύκλο ρολογιού καθυστέρηση.
* Έναν καταχωρητή μεταξύ του control unit και του mac, για να πετύχουμε τον συγχρονισμό των δεδομένων εισόδου του mac με το σήμα αρχικοποίησης mac\_init. Η καθυστέρηση αυτή είναι απαραίτητη λαμβάνοντας υπ’ όψη ότι μεταξύ της παραγωγής που σήματος διεύθυνσης και της άφιξης των δεδομένων των μνημών στο mac μεσολαβεί ένας κύκλος ρολογιού.
* Ένα Pipeline 8 καταχωρητών μέσω των οποίων μεταδίδεται το σήμα valid\_in και έπειτα από ακριβώς 8 κύκλους καταφθάνει στην έξοδο ως valid\_out.

To τελευταίο δεν είναι απαραίτητο για την ζητούμενη υλοποίηση αν δεχτούμε ότι η είσοδος είναι πάντα έτοιμη ανά 8 κύκλους. Ωστόσο, με αυτόν τον τρόπο όπως προαναφέραμε στην υλοποίηση του control unit, εξασφαλίζουμε ότι θα έχουμε έγκυρα δεδομένα στην έξοδο ακόμα και αν η είσοδος φτάσει με καθυστέρηση. Η περιγραφή του module σε VHDL καθώς και το RTL schematic φαίνονται παρακάτω:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity fir\_filter is

port(

clk: in std\_logic;

rst: in std\_logic;

valid\_in: in std\_logic;

x: in std\_logic\_vector(7 downto 0);

valid\_out: out std\_logic ;

y: out std\_logic\_vector(18 downto 0)

);

end fir\_filter;

architecture structural of fir\_filter is

component control\_unit is

port(

clk: in std\_logic;

rst: in std\_logic;

valid\_in: in std\_logic;

rom\_addr: out std\_logic\_vector(2 downto 0);

ram\_addr: out std\_logic\_vector(2 downto 0);

mac\_init: out std\_logic;

we: out std\_logic;

valid\_out: out std\_logic

);

end component;

component mlab\_ram is

port (

rst : in std\_logic;

clk : in std\_logic;

we : in std\_logic; --- memory write enable

en : in std\_logic; --- operation enable

addr : in std\_logic\_vector(2 downto 0); -- memory address

di : in std\_logic\_vector(7 downto 0); -- input data

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

); -- output data

end component;

component mlab\_rom is

Port (

clk : in STD\_LOGIC;

en : in STD\_LOGIC; --- operation enable

addr : in STD\_LOGIC\_VECTOR (2 downto 0); -- memory address

rom\_out : out STD\_LOGIC\_VECTOR (7 downto 0)

); -- output data

end component;

component mac is

port(

init: in std\_logic;

b: in std\_logic\_vector(7 downto 0);

c: in std\_logic\_vector(7 downto 0);

clk: in std\_logic;

y: out std\_logic\_vector(18 downto 0) -- 2X + 3 bits

);

end component;

signal rom\_addr\_s, ram\_addr\_s: std\_logic\_vector(2 downto 0);

signal mac\_init\_s, r\_mac\_init, valid\_out\_s, we\_s: std\_logic;

signal r\_x, rom\_out\_s, ram\_out\_s: std\_logic\_vector(7 downto 0);

signal r\_valid\_out: std\_logic\_vector(8 downto 0);

begin

cu: control\_unit port map(

clk => clk,

rst => rst,

valid\_in => valid\_in,

rom\_addr => rom\_addr\_s,

ram\_addr => ram\_addr\_s,

mac\_init => mac\_init\_s,

we => we\_s,

valid\_out => valid\_out\_s

);

rom: mlab\_rom port map(

clk => clk,

en => '1',

addr => rom\_addr\_s,

rom\_out => rom\_out\_s

);

ram: mlab\_ram port map(

rst => rst,

clk => clk,

we => we\_s,

en => '1',

addr => ram\_addr\_s,

di => r\_x,

do => ram\_out\_s

);

comp\_mac: mac port map(

init => r\_mac\_init,

b => rom\_out\_s,

c => ram\_out\_s,

clk => clk,

y => y

);

valid\_out <= r\_valid\_out(8);

sync: process(clk,rst)

begin

if rst = '1' then

r\_x <= (others => '0');

r\_mac\_init <= '0';

r\_valid\_out <= (others => '0');

else

if clk'event and clk='1' then

r\_x <= x;

r\_mac\_init <= mac\_init\_s;

r\_valid\_out(8 downto 1) <= r\_valid\_out(7 downto 0);

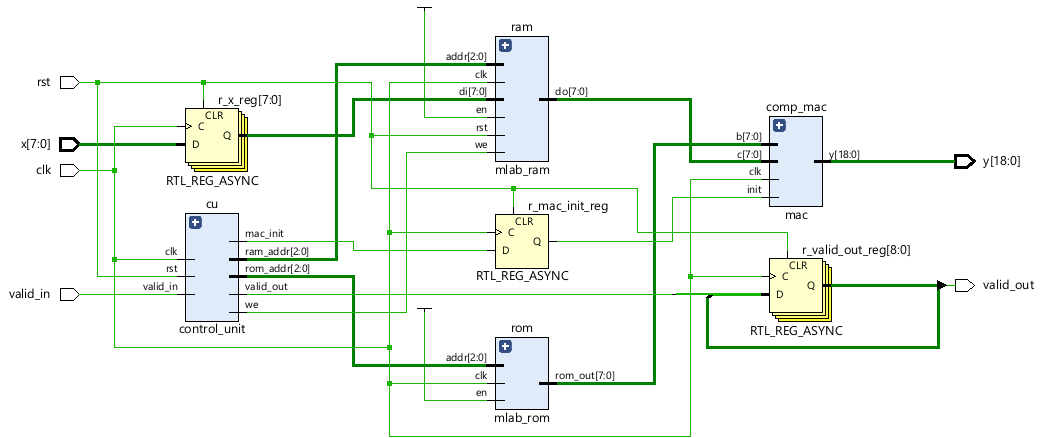
r\_valid\_out(0) <= valid\_out\_s;

end if;

end if;

end process;

end structural;



Για τη δοκιμή του κυκλώματός μας χρησιμοποιήσαμε για είσοδο 16 τυχαίες τιμές των 8 bit. Για τον εύκολο υπολογισμό των ορθών τιμών εξόδου γράψαμε έναν σύντομο script σε python:

h = [1,2,3,4,5,6,7,8]

x = [213, 107, 172, 58, 147, 225, 92, 39, 205, 26, 180, 99, 248, 15, 134, 73]

y = [0 for \_ in x]

for i in range(len(x)):

temp = i - 8

if temp < 0: temp = -1

for j in range(i,temp,-1):

y[i] += x[j] \* h[(i - j) % 8]

print(y[i])

Έτσι βρήκαμε ότι οι τιμές εξόδου του φίλτρου για τη συγκεκριμένη είσοδο είναι οι παρακάτω:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 213 | 533 | 1025 | 1575 | 2272 | 3194 | 4208 | 5261 | 4602 | 4710 | 4306 | 4855 | 4793 | 3897 | 4107 | 4775 |

Στην συνέχεια δημιουργήσαμε κατάλληλο testbench 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;

use ieee.numeric\_std.all;

entity fir\_tb is

end fir\_tb;

architecture fir\_tb\_arch of fir\_tb is

component fir\_filter

port(

clk: in std\_logic;

rst: in std\_logic;

valid\_in: in std\_logic;

valid\_out: out std\_logic;

x: in std\_logic\_vector(7 downto 0);

y: out std\_logic\_vector(18 downto 0)

);

end component;

constant CLKP : time := 1ns;

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

signal y\_s: std\_logic\_vector(18 downto 0);

signal rst\_s, clk, valid\_in\_s, valid\_out\_s: std\_logic;

type x\_array is array (0 to 15) of std\_logic\_vector(7 downto 0);

signal data : x\_array := (

"11010101", -- Random binary values

"01101011",

"10101100",

"00111010",

"10010011",

"11100001",

"01011100",

"00100111",

"11001101",

"00011010",

"10110100",

"01100011",

"11111000",

"00001111",

"10000110",

"01001001"

);

begin

clk\_proc: process

begin

clk <= '0';

wait for CLKP/2;

clk <= '1';

wait for CLKP/2;

end process;

UUT: fir\_filter port map (clk,rst\_s,valid\_in\_s,valid\_out\_s,x\_s,y\_s);

testSequence: process

begin

valid\_in\_s <= '0';

rst\_s <= '1';

wait for CLKP;

rst\_s <= '0';

for i in 0 to 15 loop

for j in 0 to 7 loop

if j=0 then

valid\_in\_s <= '1';

x\_s <= data(i);

else

valid\_in\_s <= '0';

end if;

wait for CLKP;

end loop;

end loop;

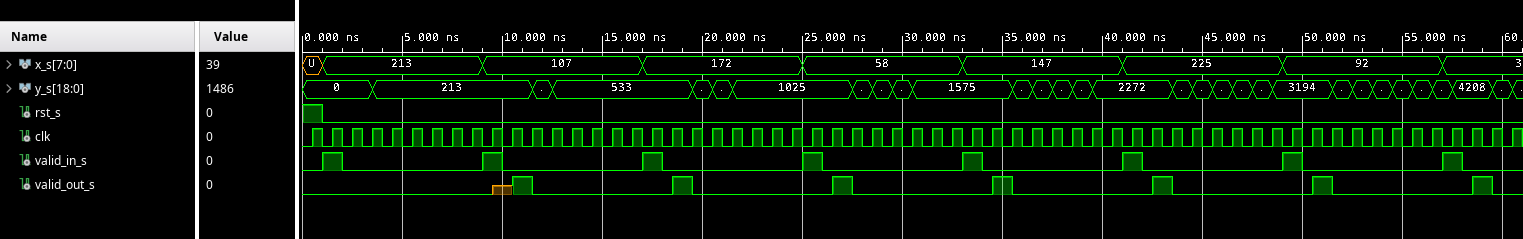
wait for 8\*CLKP;

rst\_s <= '1';

wait for CLKP;

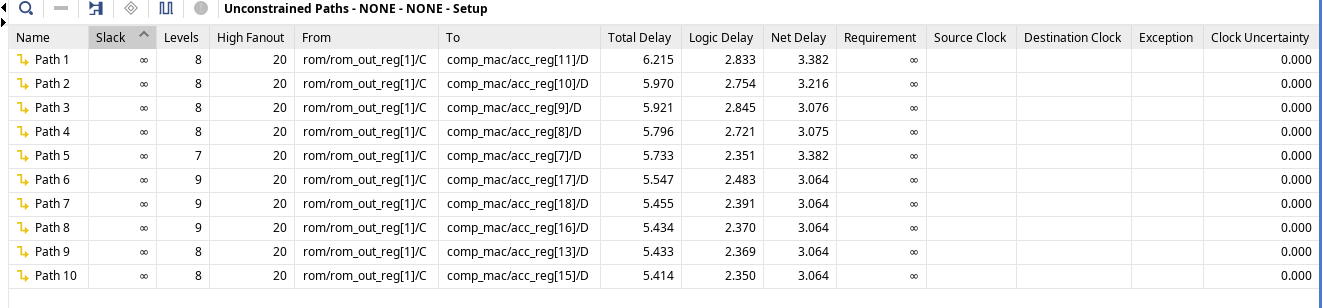
end process;

end fir\_tb\_arch;



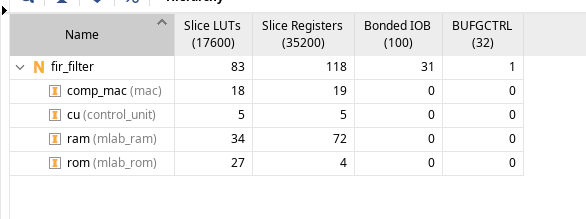
Αφού επιβεβαιώσαμε την ορθότητα της λειτουργίας του κυκλώματος με Behavioral simulation μπορούμε να προχωρήσουμε σε synthesis και implementation στο επιλεγμένο board.

Έτσι μπορούμε να δούμε την καθυστέρηση σε κάθε μέρος του κυκλώματος :



Παρατηρούμε ότι το critical path βρίσκεται μεταξύ της εξόδου της μνήμης ROM και της εισόδου του mac και ο απαιτούμενος χρόνος του είναι 6.2 ns το οποίο σημαίνει ότι το κύκλωμα αυτό μπορεί θεωρητικά να λειτουργήσει σε συχνότητες έως και 160 Mhz.

Τέλος βλέπουμε το υλικό του FPGA που χρησιμοποιήθηκε κατά τη σύνθεση και το τελικό schematic:



A diagram of a computer program

AI-generated content may be incorrect.