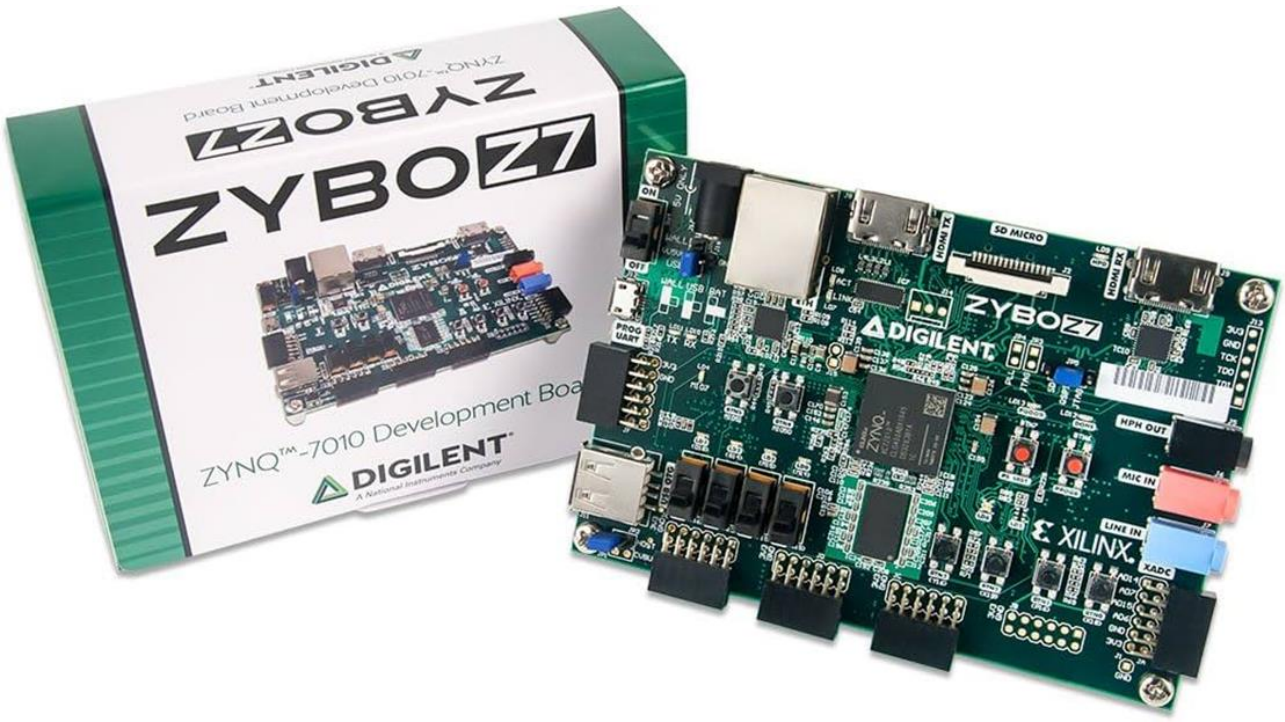
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών



Μάθημα: Ψηφιακά Συστήματα VLSI

Θέμα: 4η Εργαστηριακή Άσκηση – Υλοποίηση FIR Φίλτρου

Εξάμηνο: 8ο

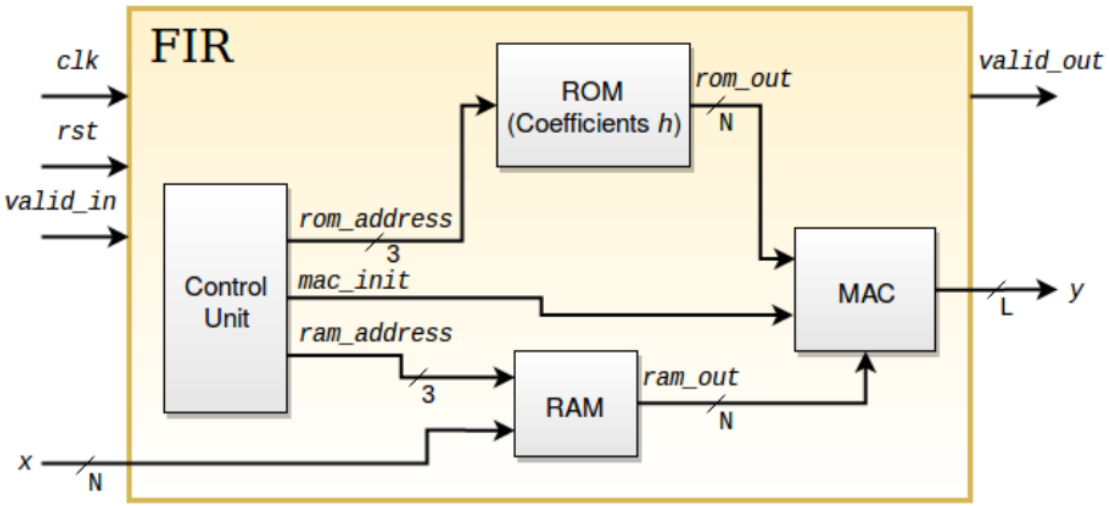
Ομάδα: 6

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:

$$y[n] = \sum_{k=0}^{7} h[k]x[n-k] = h[0]x[n] + h[1]x[n-1] + \cdots + h[7]x[n-7]$$



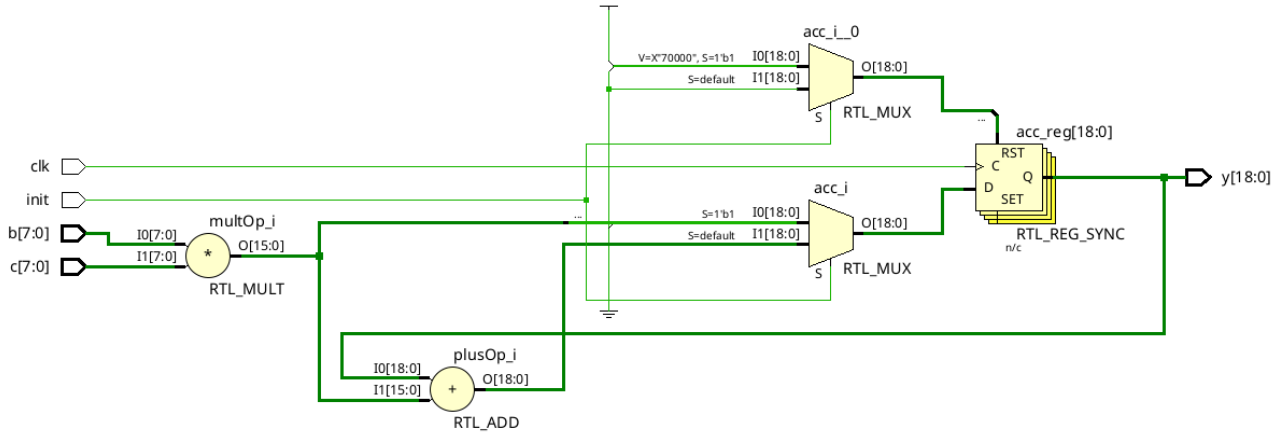
Συνεπώς το ζητούμενο της άσκησης είναι η υλοποίηση καθενός από τα 4 modules που φαίνονται στο σχεδιάγραμμα καθώς και ο συγχρονισμός τους, ώστε να έχουμε ένα λειτουργικό φίλτρο.

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');
```

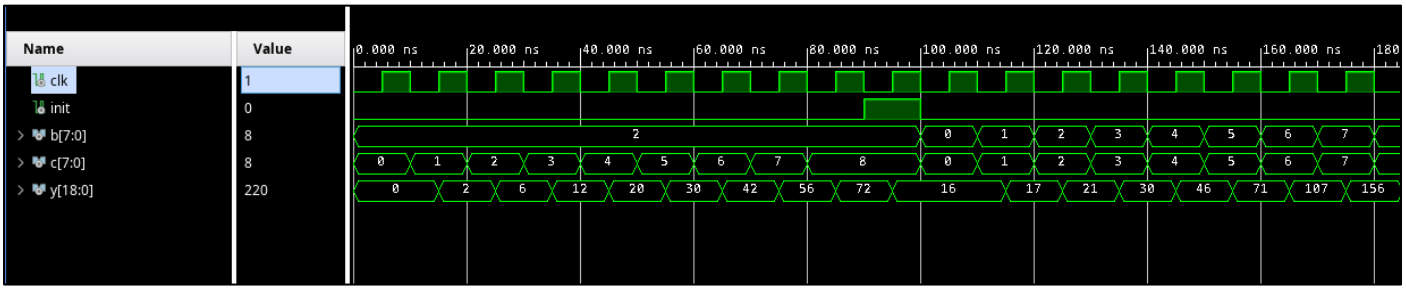
```
                acc(15 downto 0) <= b * c;
            else
                acc <= acc + (b * c);
            end if;
        end if;
end process;
        y <= acc;
end behavioral;
```



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;
```

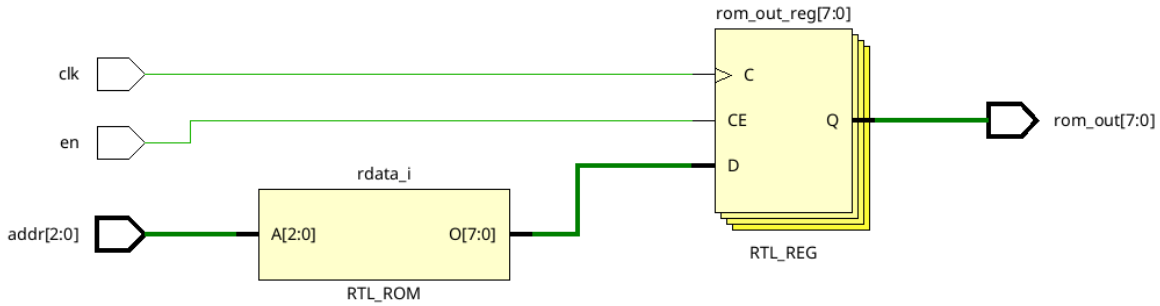


Read Only Memory (ROM)

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;
```

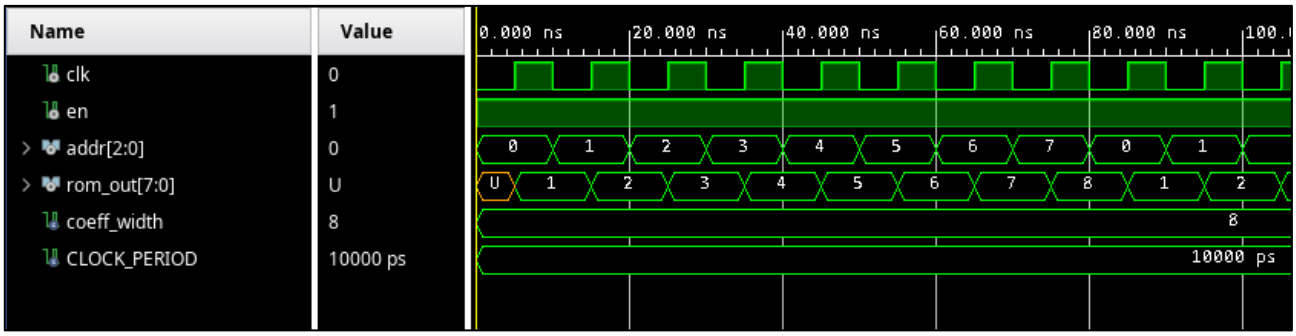
```
architecture Behavioral of mlab_rom is
      type rom_type is array (7 downto 0) of std_logic_vector (coeff_width-1 downto 0);  --
initialization of rom with user data
      signal rom : rom_type:= ("00001000", "00000111", "00000110", "00000101", "00000100",
                                    "00000011", "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;
```



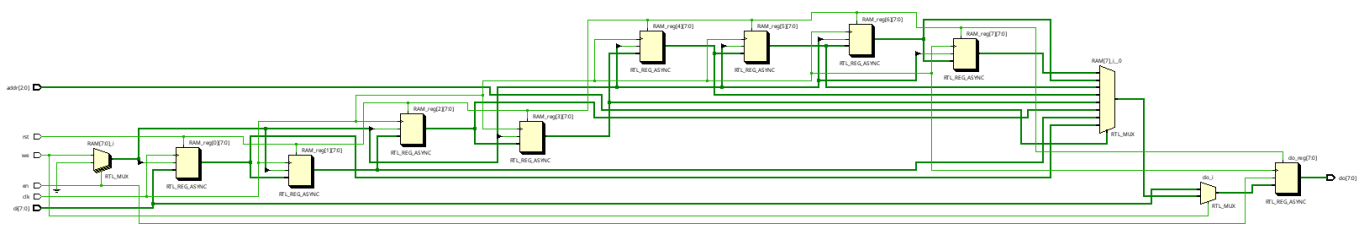
Όπως βλέπουμε παραπάνω, η ROM επιστρέφει τις τιμές που είναι αποθηκευμένες στην επιθυμητή διεύθυνση σε κάθε θετική ακμή του ρολογιού.

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:

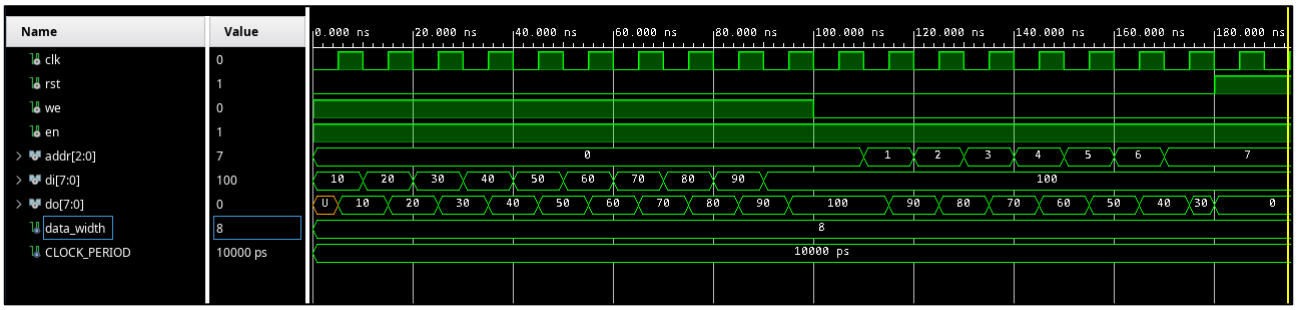
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

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;
```



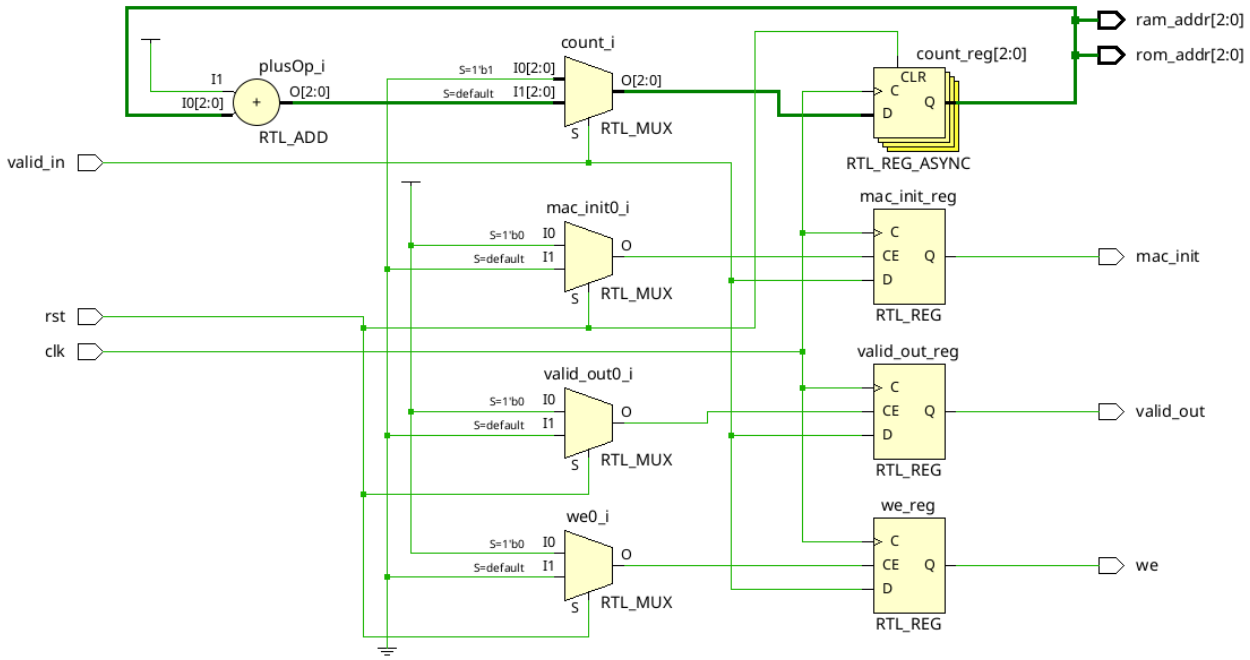
Όπως βλέπουμε και παραπάνω όταν we = '1' τα νέα δεδομένα από τη di εγγράφονται στη μνήμη, ενώ όταν we = '0' η λειτουργία της RAM ταυτίζεται με αυτή της ROM.

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
```

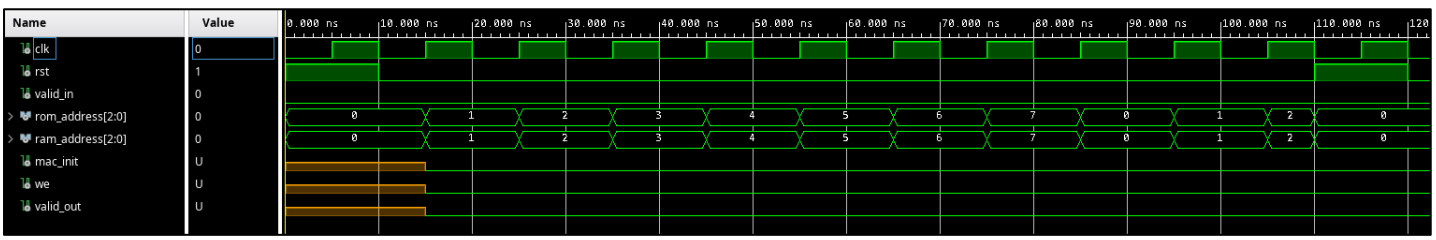
```
	if rst = '1' then
		count <= "000";
	else
		if clk'event and clk = '1' then
			if valid_in = '1' then
				count <= "000";
				valid_out <= '1';
				mac_init <= '1';
				we <= '1';
			else
				count <= count + 1;
				valid_out <= '0';
				mac_init <= '0';
				we <= '0';
			end if;
		end if;
	end 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:

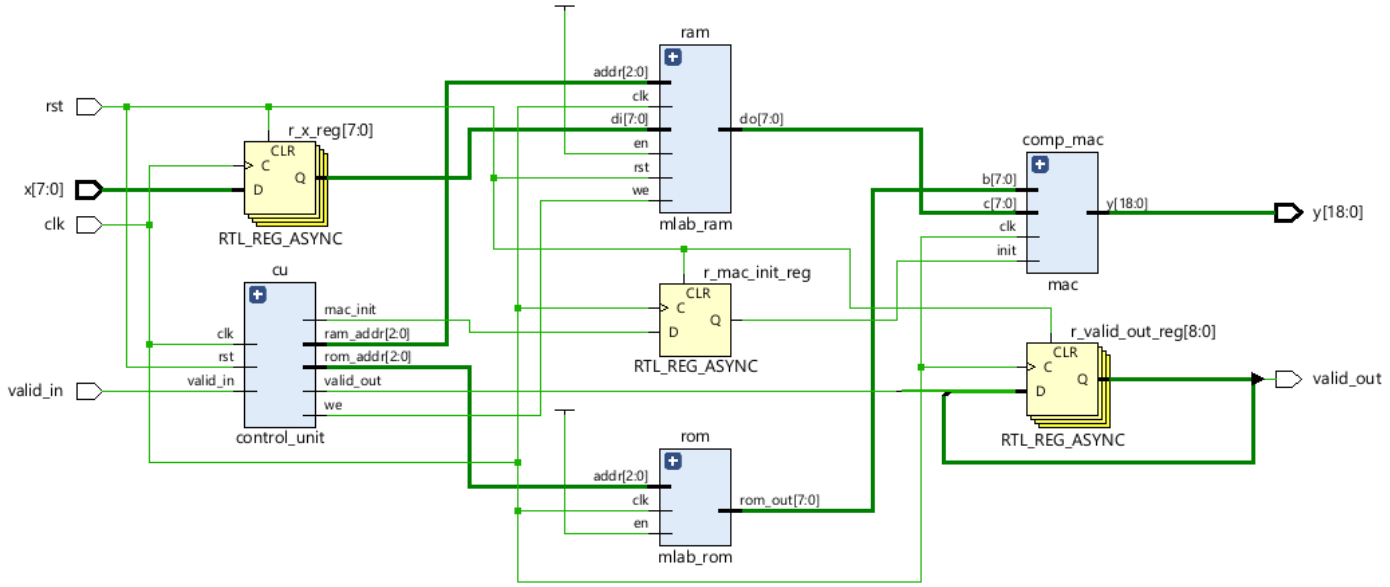
- Έναν καταχωρητή μεταξύ του σήματος εισόδου x και της εισόδου της RAM έτσι ώστε τα εισερχόμενα δεδομένα να φθάνουν ταυτόχρονα με το σήμα valid_in το οποίο μεταδίδεται μέσω του control unit με έναν κύκλο ρολογιού καθυστέρηση.
- Έναν καταχωρητή μεταξύ του control unit και του mac, για να πετύχουμε τον 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.

Το 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:

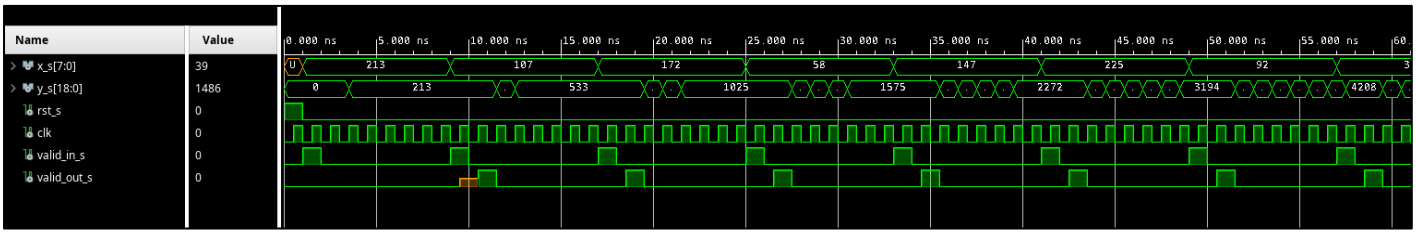
213 533 1025 1575 2272 3194 4208 5261 4602 4710 4306 4855 4793 3897 4107 4775

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 :

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	8	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[11]/D	6.215	2.833	3.382	∞				0.000
Path 2	∞	8	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[10]/D	5.970	2.754	3.216	∞				0.000
Path 3	∞	8	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[9]/D	5.921	2.845	3.076	∞				0.000
Path 4	∞	8	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[8]/D	5.796	2.721	3.075	∞				0.000
Path 5	∞	7	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[7]/D	5.733	2.351	3.382	∞				0.000
Path 6	∞	9	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[17]/D	5.547	2.483	3.064	∞				0.000
Path 7	∞	9	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[18]/D	5.455	2.391	3.064	∞				0.000
Path 8	∞	9	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[16]/D	5.434	2.370	3.064	∞				0.000
Path 9	∞	8	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[13]/D	5.433	2.369	3.064	∞				0.000
Path 10	∞	8	20	rom/rom_out_reg[1]/C	comp_mac/acc_reg[15]/D	5.414	2.350	3.064	∞				0.000

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:

Name	Slice LUTs (17600)	Slice Registers (35200)	Bonded IOB (100)	BUFGCTRL (32)
N fir_filter	83	118	31	1
comp_mac (mac)	18	19	0	0
cu (control_unit)	5	5	0	0
ram (mlab_ram)	34	72	0	0
rom (mlab_rom)	27	4	0	0