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

4η 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

Παρακάτω ακολουθεί κώδικας και ανάλυση για καθένα από τα βασικά δομικά στοιχεία του FIR φίλτρου:

- Control Unit: Η βασική μονάδα που ελέγχει την λειτουργία του φίλτρου. Συγκεκριμένα, ανάλογα με την τιμή της εισόδου valid_in και το κατά πόσο έχει ολοκληρωθεί ο ήδη υπάρχον υπολογισμός από τον MAC, ορίζει τις τιμές των σημάτων valid_out & mac_init. Σε περίπτωση που έρθει είσοδος πριν τους 8 κύκλους που απαιτούνται για την ολοκλήρωση της πράξης, το control unit περιμένει μέχρι την ολοκλήρωσή της. Σε περίπτωση που καθυστερήσει η επόμενη είσοδος, τότε το φίλτρο μπαίνει σε κατάσταση αναμονής μέχρι να έρθει η επόμενη τιμή X με valid_in=1.

```
entity Control_Unit is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           valid_in : in STD_logic;
           mac_init: out STD_LOGIC;
           ram_addr : out STD_LOGIC_VECTOR (2 downto 0);
           rom_addr : out STD_LOGIC_VECTOR (2 downto 0);
           counter: out STD_LOGIC_VECTOR (2 downto 0);
           we: out STD_LOGIC;  --to we twn RAM/ROM
           valid_out: out STD_LOGIC
           );
end entity Control_Unit;

architecture Behavioral of Control_Unit is
    signal count_reg : STD_LOGIC_VECTOR (2 downto 0) := (others =>
'1');
    signal flag, waitt : std_logic ;

begin
    process (clk)
    begin
       if rst = '1' then
          count_reg <= (others => '0');
          valid_out<='0';
          mac_init<='1';
       elsif rising_edge(clk) then
         if waitt='1' then
           valid_out <= '0';
           if valid_in='1' then --stop the wait condition
               we<='1';
               mac_init<='1';
               waitt<='0';
           end if;
         else
       if count_reg=0 then --new ouput
           valid_out<='1';
           if valid_in='1' then --start new calculation
               we<='1';
               mac_init<='1'; --initialize mac
           else
               waitt<='1'; --not valid input (wait again)
```

```
            end if;
        else --still calculating
            valid_out<='0';
            we<='0';
            mac_init<='0';
        end if;
    end if;
    count_reg <= count_reg + 1;
    ram_addr <= count_reg;
   rom_addr <= count_reg;
   counter <= count_reg;
   end if;
   end process;
end architecture;
```

- MAC: Η βασική μονάδα που υπολογίζει την έξοδο του φίλτρου y, πολλαπλασιάζοντας κάθε συντελεστή του φίλτρου (έξοδος της ROM) με το αντίστοιχο σήμα εισόδου (έξοδος της RAM). Επιπλέον, αρχικοποιεί εκ νέου την έξοδο του κάθε φορά που ισχύει mac_init=1, και για τους επόμενους 7 κύκλους ρολογιού προσθέτει το γινόμενο των εξόδων της RAM και της ROM στο ήδη υπάρχον αποτέλεσμα. Αξίζει να σημειωθεί ότι για να αποφευχθεί το φαινόμενο υπερχείλισης της εξόδου λόγω πολλαπλασιασμού με 2 8-bit αριθμούς οι οποίοι προστίθενται κάθε φορά σε έναν τρίτο 8-bit αριθμό, ορίστηκε η έξοδος y να έχει μέγεθος length_ram_out + length_rom_out + ceil(log(M)) = 8+8+3=19 bits.

```
entity MAC is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           rom_out : in STD_LOGIC_VECTOR (7 downto 0);
           ram_out : in STD_LOGIC_VECTOR (7 downto 0);
           mac_init : in STD_LOGIC;
           y : out STD_LOGIC_VECTOR (18 downto 0));
end entity MAC;

architecture Behavioral of MAC is
    signal  sum : STD_LOGIC_VECTOR (18 downto 0) := (others => '0');

begin
    process (clk)
    begin
     if(rst = '1') then
      sum <= (others => '0');
       elsif rising_edge(clk) then
            if mac_init = '1' then
               sum <= (others => '0'); --Asing all '0's to my vector
               sum(15 downto 0) <= rom_out*ram_out;
            else
               sum <=sum + rom_out*ram_out; -- Perform multiplication
-- and addition;
         end if;
         y <= sum;
```

```
        end if;
    end process;
end architecture;
```

- RAM: Η βασική μονάδα που αποθηκεύει την παρούσα τιμή της εισόδου X, καθώς και τις 7 προηγούμενές της. Ανάλογα με την τιμή ram_address που δέχεται, δίνει και την αντίστοιχη έξοδο ram_out στην μονάδα MAC. Η επιλογή read/write γίνεται από το control unit, με το write να συμπίπτει με ανάθεση του mac_init σε 1, μιας και τότε έχουμε νέα έγκυρη τιμή εισόδου προς υπολογισμό, η οποία μπορεί να γραφτεί και στην RAM.

```
entity mlab_ram is
    generic (
        data_width : integer :=8                    --- width of
data (bits)
    );
    port (clk  : in std_logic;
          rst  : 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)
    begin
        if rst='1' then
             ram <= (others=>(others=>'0'));
        elsif clk'event and clk = '1' then
            if en = '1' then
                if we = '1' then                    -- write
operation
                   for i in 0 to 6 loop
                        RAM(7-i) <= RAM(6-i);
                   end loop;
                    RAM(0) <= di;
                    do <= di;
                else                                -- read
operation
                   do <= RAM( conv_integer(addr));
```

```
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

- ROM: Η βασική μονάδα που έχει αποθηκευμένες τους 8 συντελεστές του φίλτρου. Σε αντιστοιχία με την μνήμη RAM, ανάλογα με την τιμή rom_address που δέχεται από το control unit, δίνει και την αντίστοιχη έξοδο rom_out στην μονάδα MAC. Η επιλογή read/write γίνεται από το control unit, και ταυτίζεται με αυτήν για την μνήμη RAM.

```
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);
                              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)
    begin
        if (clk'event and clk = '1') then
            if (en = '1') then
                rom_out <= rdata;
            end if;
        end if;
    end process;
end Behavioral;
```

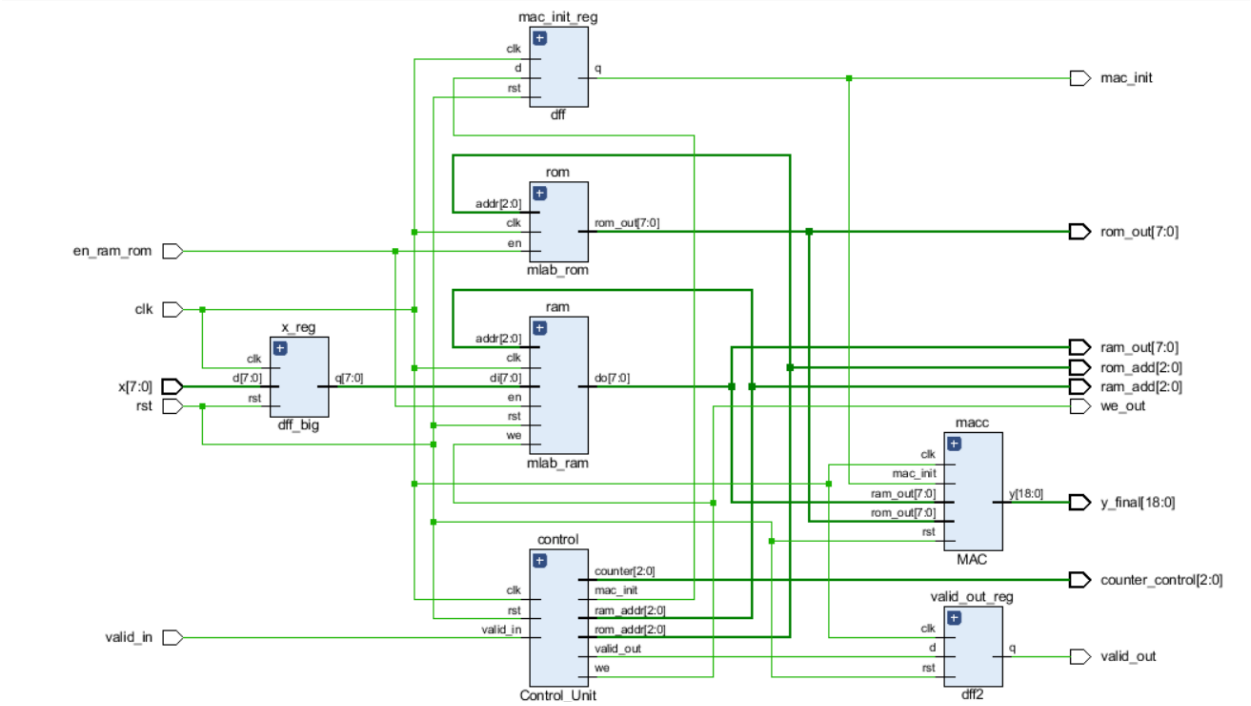
Με βάση τα άνωθι components, υλοποιήθηκε το ζητούμενο FIR filter entity. Αξίζει να τονισθεί ότι, για κατάλληλο συγχρονισμό των αποτελεσμάτων, τα σήματα X και mac_init περνούν από ένα D Flip-Flop (καθυστέρηση 1 κύκλου ρολογιού), ενώ το σήμα valid_out από 2 D Flip-Flop (καθυστέρηση 2 κύκλων ρολογιού). Ο κώδικας του φίλτρου ακολουθεί παρακάτω:

```
entity FIR is
  Port ( clk : in std_logic;
         rst : in std_logic;
         valid_in : in std_logic;
         en_ram_rom: in std_logic;
         x : in std_logic_vector(7 downto 0);
         valid_out : out std_logic;
         y_final : out std_logic_vector (18 downto 0);
         rom_out,ram_out:out STD_LOGIC_VECTOR (7 downto 0);
         rom_add,ram_add,counter_control:out STD_LOGIC_VECTOR (2
downto 0);
         mac_init : out std_logic;
         we_out:out std_logic
         );
end FIR;

architecture Behavioral of FIR is
    --comp1
    component 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 component;

    --comp2
    component mlab_ram is
         generic (
            data_width : integer :=8                  --- width of
data (bits)
         );
        port (clk  : in std_logic;
              rst  : in std_logic;
              we   : in std_logic;                        --- memory
write enable
              en   : in std_logic;                --- operation enable
```

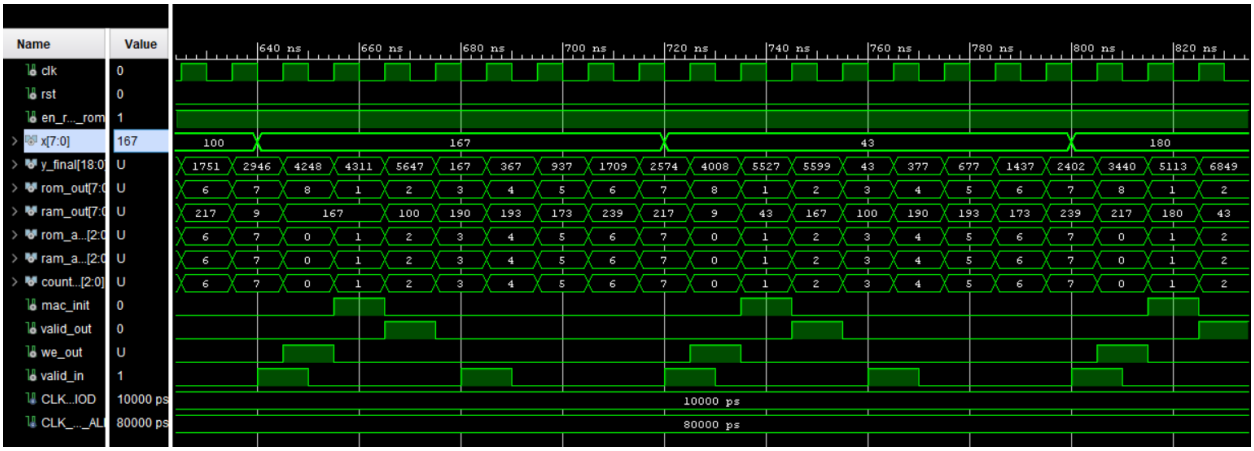
```
              addr : in std_logic_vector(2 downto 0);            --
memory address
              di   : in std_logic_vector(data_width-1 downto 0);
-- input data
              do   : out std_logic_vector(data_width-1 downto 0));
-- output data
    end component;

    --comp3
    component Control_Unit is
        Port ( clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               valid_in : in STD_logic;
               mac_init: out STD_LOGIC;
               ram_addr : out STD_LOGIC_VECTOR (2 downto 0);
               rom_addr : out STD_LOGIC_VECTOR (2 downto 0);
               we: out STD_LOGIC; --to we twn RAM/ROM
               counter: out STD_LOGIC_VECTOR (2 downto 0);
               valid_out: out STD_LOGIC
               );
    end component;

    --comp4
    component MAC is
        Port ( clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               rom_out : in STD_LOGIC_VECTOR (7 downto 0);
               ram_out : in STD_LOGIC_VECTOR (7 downto 0);
               mac_init : in STD_LOGIC;
--               valid_out: in STD_LOGIC;
               y : out STD_LOGIC_VECTOR (18 downto 0));
    end component;

    --comp5
    component dff is
     port(
          d : in  std_logic;
          q : out  std_logic;
          clk : in std_logic;
          rst : in std_logic
         );
    end component;

    --comp6
    component dff_big is
        Port ( d : in STD_LOGIC_VECTOR (7 downto 0);
               clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               q : out STD_LOGIC_VECTOR (7 downto 0));
    end component;

    --comp7
```

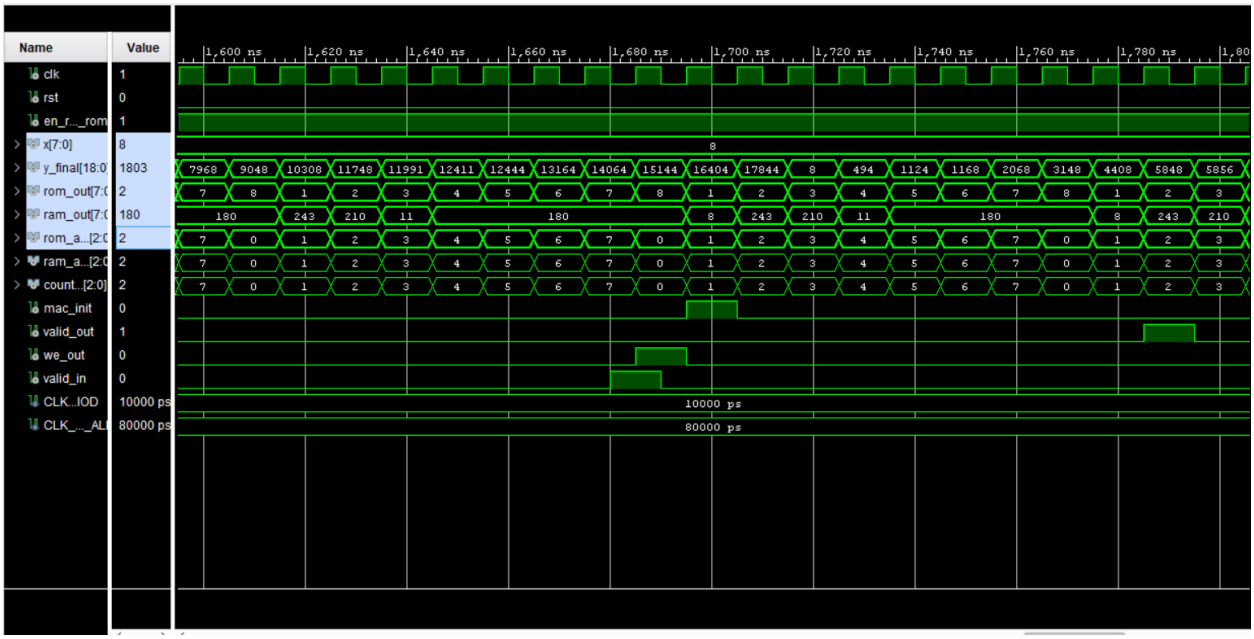
```
    component dff2 is
        port(
            d : in  std_logic;
            q : out  std_logic;
            clk : in std_logic;
            rst : in std_logic
           );
    end component;

    signal x_dff,ram_out_temp,rom_out_temp : std_logic_vector(7 downto
0);
    signal ram_addr_temp, rom_addr_temp,counter_temp:
std_logic_vector(2 downto 0);
    signal mac_init_temp,
we_temp,valid_out_temp,mac_init_dff,valid_out_dff: std_logic;
    begin
    x_reg: dff_big port map(d=>x, clk=>clk, rst=>rst, q=>x_dff);
    control: Control_Unit port map(clk=>clk, rst=>rst,
valid_in=>valid_in, mac_init=>mac_init_temp, ram_addr=>ram_addr_temp,
rom_addr=>rom_addr_temp,we=>we_temp,valid_out=>valid_out_temp,counter=
>counter_temp);
--    control: Control_Unit port map(clk=>clk, rst=>rst,
mac_init=>mac_init_temp, ram_addr=>ram_addr_temp,
rom_addr=>rom_addr_temp,we=>we_temp);
    mac_init_reg : dff port map(d=>mac_init_temp, clk=>clk, rst=>rst,
q=>mac_init_dff);
    valid_out_reg : dff2 port map(d=>valid_out_temp, clk=>clk,
rst=>rst, q=>valid_out_dff);
    rom: mlab_rom port map(clk=>clk,en=>en_ram_rom,
addr=>rom_addr_temp,rom_out=>rom_out_temp);
    ram: mlab_ram port map(clk=>clk, rst=>rst, en=>en_ram_rom,
we=>we_temp ,addr=>ram_addr_temp,di=>x_dff, do=>ram_out_temp);
    macc: MAC port map(rst=>rst,clk=>clk, rom_out=>rom_out_temp,
ram_out=>ram_out_temp, mac_init=>mac_init_dff, y=>y_final);

    --just for simulation
   --RAM_IN <= x_dff;
   mac_init <=mac_init_dff;
   we_out <= we_temp;
   ram_out <=ram_out_temp;
   rom_out <=rom_out_temp;
   rom_add <=rom_addr_temp;
   ram_add <=ram_addr_temp;
   counter_control <= counter_temp;
   valid_out <=valid_out_dff;
   end Behavioral;
```

Το RTL Schematic για το FIR φίλτρο είναι το εξής:



Στην συνέχεια, γράφτηκε ένα testbench για να επαληθευτεί η ορθή λειτουργία του φίλτρου, με τα αποτελέσματα να ακολουθούν παρακάτω





Παρατηρούμε ότι-τόσο στην περίπτωση που έρχεται είσοδος νωρίτερα από 8 κύκλους ρολογιού όσο και στην περίπτωση που έρχεται αργότερα-το φίλτρο εξάγει τα επιθυμητά αποτελέσματα: στην $1^{η}$ περίπτωση, περιμένει να ολοκληρωθεί ο προηγούμενος υπολογισμός και ορθώς δεν δέχεται την ενδιάμεση είσοδο (τα mac_init και we_out παραμένουν στο 0). Στην $2^{η}$ περίπτωση, παρατηρούμε ότι όσο δεν υπάρχει νέα valid τιμή εισόδου η ram_out τιμή παραμένει σταθερή στην τελευταία valid τιμή, ενώ όταν ξανατίθεται το valid_in σε 1, τότε γίνεται εκ νέου υπολογισμός από το φίλτρο η έξοδος του οποίου είναι έγκυρη μετά από 8 κύκλους ρολογιού.

Τέλος, παρατίθενται αποτελέσματα που αφορούν το μέγιστο μονοπάτι καθυστέρησης και την κατανάλωση του φίλτρου:

General Information
Timer Settings
Design Timing Summary
Check Timing (518)

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source C
Path 1	∞	10	10	20	rom/rom_out_reg[1]/C	macc/sum_reg[17]/D	5.617	2.703	2.914	∞	
Path 2	∞	10	10	20	rom/rom_out_reg[1]/C	macc/sum_reg[18]/D	5.525	2.611	2.914	∞	
Path 3	∞	10	10	20	rom/rom_out_reg[1]/C	macc/sum_reg[16]/D	5.504	2.590	2.914	∞	

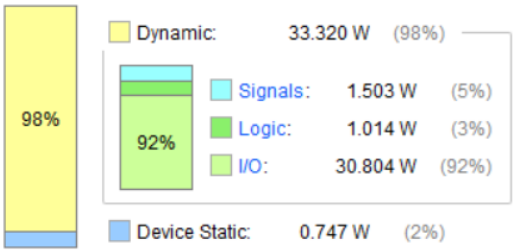
Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**34.068 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-332.9°C (-28.3 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



Ως συμπέρασμα μπορεί να εξαχθεί ότι το φίλτρο που υλοποιήθηκε, αν και ικανοποιητικά γρήγορο λόγω του χαμηλού του delay (5.617ns), έχει ιδιαίτερα υψηλή κατανάλωση και υπερβαίνει τα επιθυμητά όρια θερμοκρασίας. Ένας λόγος που μπορεί αυτό να συμβαίνει έχει να κάνει με το γεγονός ότι όλα τα components του φίλτρου χρησιμοποιούν το ρολόι εισόδου κάτι το οποίο δεν είναι αναγκαίο και θα μπορούσε να αποτραπεί, ιδιαίτερα στην περίπτωση του mac unit, η αρχικοποίηση του οποίου εξαρτάται μόνο από το mac_init του control unit.