Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

4η Σειρά Ασκήσεων

Μάθημα: Ψηφιακά Συστήματα VLSI

Εξάμηνο: 8ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Υλοποίηση FIR Φίλτρου

Σε αυτή την εργαστηριακή άσκηση, θα ασχοληθούμε με την υλοποίηση ενός FIR φίλτρου. Τα φίλτρα FIR (Finite Impulse Response) είναι φίλτρα των οποίων η παλμική απόκριση (ή απόκριση σε οποιαδήποτε είσοδο πεπερασμένου μήκους) είναι πεπερασμένης διάρκειας, επειδή καθιζάνει στο μηδέν σε πεπερασμένο χρόνο.

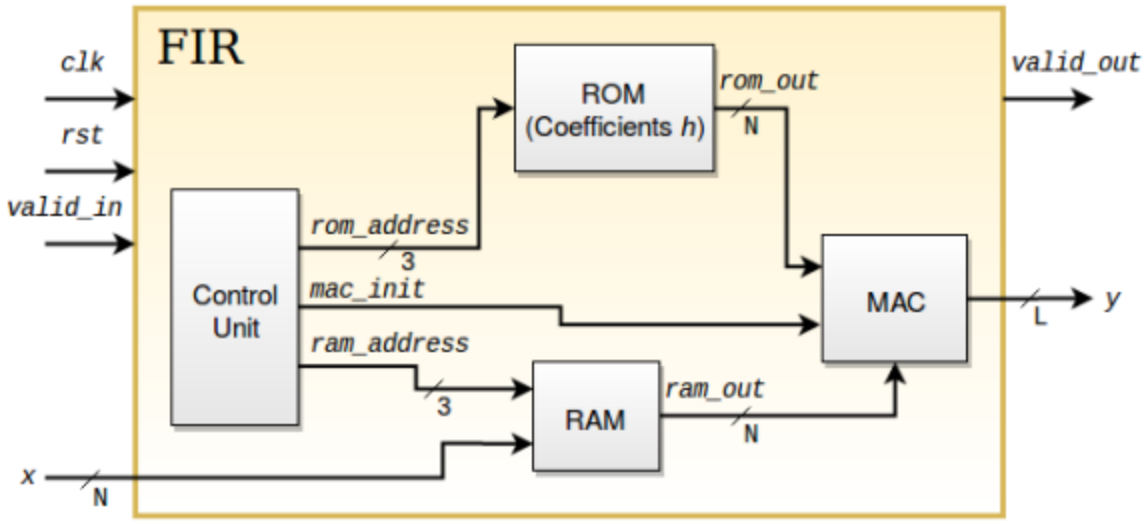
Στη γενική σχέση εισόδου - εξόδου ενός FIR φίλτρου είναι η ακόλουθη:

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

Όπου:

- M είναι η τάξη του φίλτρου
- $y[n]$ είναι η έξοδος του φίλτρου την διακριτή χρονική στιγμή n
- $h[k]$ είναι ο k-οστός συντελεστής του φίλτρου, με k =0, 1, 2, …, M
- $x[n]$ είναι η τιμή του σήματος εισόδου τη διακριτή χρονική στιγμή n

Στα πλαίσια της εργαστηριακής άσκησης θα υλοποιήσουμε ένα 8-tap FIR φίλτρο, σύμφωνα με την παρακάτω αρχιτεκτονική. Η άσκηση θα υλοποιηθεί για N = 8 bits και εύρος δεδομένων x.



Τα βασικά modules της αρχιτεκτονικής είναι τα ακόλουθα:

1. MAC - Μονάδα Πολλαπλασιασμού με Συσσώρευση (Multiplier Accumulator Unit):

Υπολογίζει την έξοδο του φίλτρου y, εκτελώντας την ακόλουθη πράξη: $a \leftarrow a + b \times c$
Δέχεται ως είσοδο του σταθερούς συντελεστές του φίλτρου (*rom_out*), τις τιμές του σήματος εισόδου (*ram_out*), καθώς και ένα σήμα αρχικοποίησης (*mac_init*) που υποδηλώνει την αρχικοποίηση της συσσώρευσης πριν τον υπολογισμό κάθε νέας τιμής του y.

Η υλοποίηση αυτής της μονάδας να γίνει σε behavioral περιγραφή και με τη χρήση των τελεστών '+' και '*' που υποστηρίζονται από τη βιβλιοθήκη std_logic_unsigned.

Σε κάθε κύκλο ρολογιού αθροίζει στο αποτέλεσμα που έχει ήδη αποθηκευμένο, το γινόμενο των αριθμών που λαμβάνει από RAM και ROM ώστε να σχηματίσει το τελικό αποτέλεσμα. Αν λάβει σήμα αρχικοποίησης μηδενίζει την τιμή εντός του ώστε να υπολογίσει νέο άθροισμα γινομένων.

Για να αποφύγουμε το φαινόμενο της υπερχείλισης της εξόδους λόγω του πολλαπλασιασμού των δύο 8-bit αριθμών που προστίθενται κάθε φορά σε έναν τρίτο 8-bit αριθμό, ορίζουμε την έξοδο y να έχει μέγεθος μήκος(ram_out) + μήκος(rom_out) + log(M) = 8+8+3 = 19

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
```

```
entity MAC is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           rom_output : in STD_LOGIC_VECTOR (7 downto 0);
           ram_output : 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  accum : STD_LOGIC_VECTOR (18 downto 0) := (others =>
'0');

begin
    process (clock)
    begin
     if(reset = '1') then
        accum <= (others => '0');
     elsif (clock'event and clock = '1') then
        if mac_init = '1' then
            accum <= (others => '0');
            accum(15 downto 0) <= rom_output*ram_output;
        else
            accum <=accum + rom_output*ram_output;
        end if;
         y <= accum;
     end if;
    end process;
end architecture;
```

2. ROM: έχει αποθηκευμένους τους σταθερούς συντελεστές του φίλτρου h

Δέχεται ως είσοδο μία διεύθυνση (*rom_address*) και δίνει στην έξοδο τον αντίστοιχο συντελεστή που είναι αποθηκευμένος σε αυτή (*rom_out*).

Η υλοποίηση αυτής της μονάδας (αρχικοποιημένη με τους συντελεστές) μας παρέχεται έτοιμη και αρκεί να την ενσωματώσετε κατάλληλα στη συνολική αρχιτεκτονική του φίλτρου.

```
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 ( clock : in  STD_LOGIC;
              en : in  STD_LOGIC;                      --- operation
enable
           addr : in  STD_LOGIC_VECTOR (2 downto 0);              --
memory address
           rom_output : 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 (clock)
    begin
        if (clock'event and clock = '1') then
            if (en = '1') then
                rom_output <= rdata;
            end if;
        end if;
    end process;
end Behavioral;
```

3. RAM: αποθηκεύει την παρούσα τιμή του σήματος εισόδου x, καθώς και τις 7 προηγούμενες, που είναι απαραίτητες για τον υπολογισμό της εξόδου y σύμφωνα με την εξίσωση (1).

Δέχεται ως είσοδο μία διεύθυνση (*ram_address*) και δίνει στην έξοδο την αντίστοιχη τιμή του σήματος εισόδου που είναι αποθηκευμένη σε αυτή (*ram_out*).

Κατά τη διάρκεια της λειτουργίας εγγραφής, εγγράφεται η νέα τιμή του σήματος εισόδου x στην πρώτη θέση της μνήμης και οι ήδη αποθηκευμένες 8 τιμές ολισθαίνουν κατά μία θέση με αποτέλεσμα η παλαιότερη χρονικά να διαγράφεται.

Για την υλοποίηση αυτής της μονάδας σας παρέχεται έτοιμος ο κώδικας μίας write-first μνήμης (τα νέα δεδομένα που εγγράφονται σε συγκεκριμένη διεύθυνση βγαίνουν και στην έξοδο της μνήμης στον ίδιο κύκλο ρολογιού). Βασιζόμενοι σε αυτή την μονάδα, θα πρέπει να προσθέσετε ένα μικρό κομμάτι κώδικα το οποίο θα υλοποιεί την μετατόπιση των δεδομένων της μνήμης κατά τη διάρκεια της εγγραφής σύμφωνα με την παραπάνω περιγραφή. Αρχικά όλες οι θέσεις της μνήμης είναι αρχικοποιημένες στο '0' μιας και δεν έχει γίνει καμία εγγραφή.

```
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 (clock  : in std_logic;
          reset  : 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 (clock)
    begin
        if reset='1' then
             ram <= (others=>(others=>'0'));
        elsif clock'event and clock = '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;
```

4. Control Unit - Μονάδα Ελέγχου: αποτελεί τη μονάδα που ελέγχει και καθορίζει τη λειτουργία του φίλτρου:
    - Παράγει το σήμα αρχικοποίησης του MAC (*mac_init*).
    - Διευθυνσιοδοτεί ταυτόχρονα τις μνήμες ROM και RAM για τη λειτουργία ανάγνωσης των αντίστοιχων τιμών μέσω των σημάτων *rom_address* και *ram_address* αντίστοιχα. Σε κάθε κύκλο ρολογιού δίνει την επόμενη διεύθυνση των μνημών για ανάγνωση.

Η υλοποίηση αυτής της μονάδας να βασιστεί σε έναν μετρητή (up-counter) ο οποίος θα χρησιμοποιηθεί για τη διευθυνσιοδότηση των μνημών ROM και RAM. Η υλοποίηση της μονάδας αυτής να γίνει σε behavioral περιγραφή και να χρησιμοποιηθεί η βιβλιοθήκη std_logic_unsigned για την υλοποίηση του μετρητή (χρησιμοποιώντας τον τελεστή ‘+’) .

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity Control_Unit is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           valid_in : in STD_logic;
           mac_init: out STD_LOGIC;
           ram_address : out STD_LOGIC_VECTOR (2 downto 0);
           rom_address : 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 counter : STD_LOGIC_VECTOR (2 downto 0) := (others =>
'0');
    signal  pause : std_logic ;
    signal accumulator : std_logic := '0';
    signal flag : std_logic;
    signal valid_out_temp:  STD_LOGIC;
    signal temp1,temp2,temp3,temp4,temp5,temp6 :std_logic ;


       component dff2 is
           port(
               d : in  std_logic;
               q : out  std_logic;
               clock : in std_logic;
               reset : in std_logic
              );
       end component;


begin

    process (clock)
    begin

       if reset = '1' then
```

```
      counter <= (others => '0');
      valid_out_temp<='0';
      mac_init<='1';
   elsif (clock'event and clock = '1') then
   if (valid_in = '1') and (accumulator = '0') then
       flag<='1';
       end if;
     if counter = "000" then
      mac_init<='1';
        if accumulator = '1' then
           valid_out_temp <= '1';
        else   valid_out_temp <= '0';
        end if;

        if(valid_in = '0') then
            we<= '0';
        accumulator <= '0';
        else
          we<= '1';
          accumulator <= '1';
        end if;
     else
         we <='0';
         valid_out_temp<='0';
         mac_init<='0';
     end if;

     if counter = "111" then
        flag <= '0' ;
     end if;

     if flag='1' then
        counter <= counter+1;
     else
        counter <= "000";
     end if;
ram_address <= counter;
rom_address <= counter;
```

```
  end if;
  end process;

    vout_dff1 : dff2 port map(d=>valid_out_temp, clock=>clock,
reset=>reset, q=>temp1);
    vout_dff2 : dff2 port map(d=>temp1, clock=>clock, reset=>reset,
q=>temp2);
    vout_dff3 : dff2 port map(d=>temp2, clock=>clock, reset=>reset,
q=>temp3);
    vout_dff4 : dff2 port map(d=>temp3, clock=>clock, reset=>reset,
q=>valid_out);

end architecture;
```

Για την υλοποίηση του φίλτρου FIR χρησιμοποιήσαμε, εκτός από τα παραπάνω, και τα components dff, dff2 (pipeline 2 diff) και dff_8bit (με 8-bit I/O). Παρατίθεται ο κώδικας για καθένα από αυτά:

- dff:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dff is
 port(
      d : in  std_logic;
      q : out  std_logic;
      clock : in std_logic;
      reset : in std_logic
     );
end entity;

architecture behavioural of dff is
begin
   process(clock, reset)
   begin
       if reset = '1' then
           q <= '0';
       elsif clock' event and clock = '1' then
```

```
            q <= d;
        end if;
    end process;
end behavioural;
```

- dff2:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dff2 is
    port(
        d : in  std_logic;
        q : out  std_logic;
        clock : in std_logic;
        reset : in std_logic
       );
end dff2;

architecture structural of dff2 is
    component dff is
        port(
          d : in  std_logic;
          q : out  std_logic;
          clock : in std_logic;
          reset : in std_logic
         );
end component;

signal buff : std_logic;
  begin
    delay3 : dff
    port map (
        d => d,
        q => buff,
        clock => clock,
        reset => reset
    );
```

```
    delay4 : dff
    port map (
        d => buff,
        q => q,
        clock => clock,
        reset => reset
    );
end architecture;
```

- dff_big:

```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity dff_8bit is
    Port ( d : in STD_LOGIC_VECTOR (7 downto 0);
           clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           q : out STD_LOGIC_VECTOR (7 downto 0));
end dff_8bit;

architecture Behavioral of dff_8bit is

begin
    process(clock, reset) begin
        if reset='1' then
            q<="00000000";
        elsif (clock'event and clock = '1') then
            q<=d;
        end if;
    end process;
end Behavioral;
```

Για τον κατάλληλο συγχρονισμό των εξόδων, τα σήματα x και mac_init διοχετεύονται σε ένα structure dff, ενλω το σήμα valid_in διοχετεύεται σε ένα structure dff2, για καθυστέρηση 2 κύκλων. Παρατίθεται ο κώδικας του FIR φίλτρου:

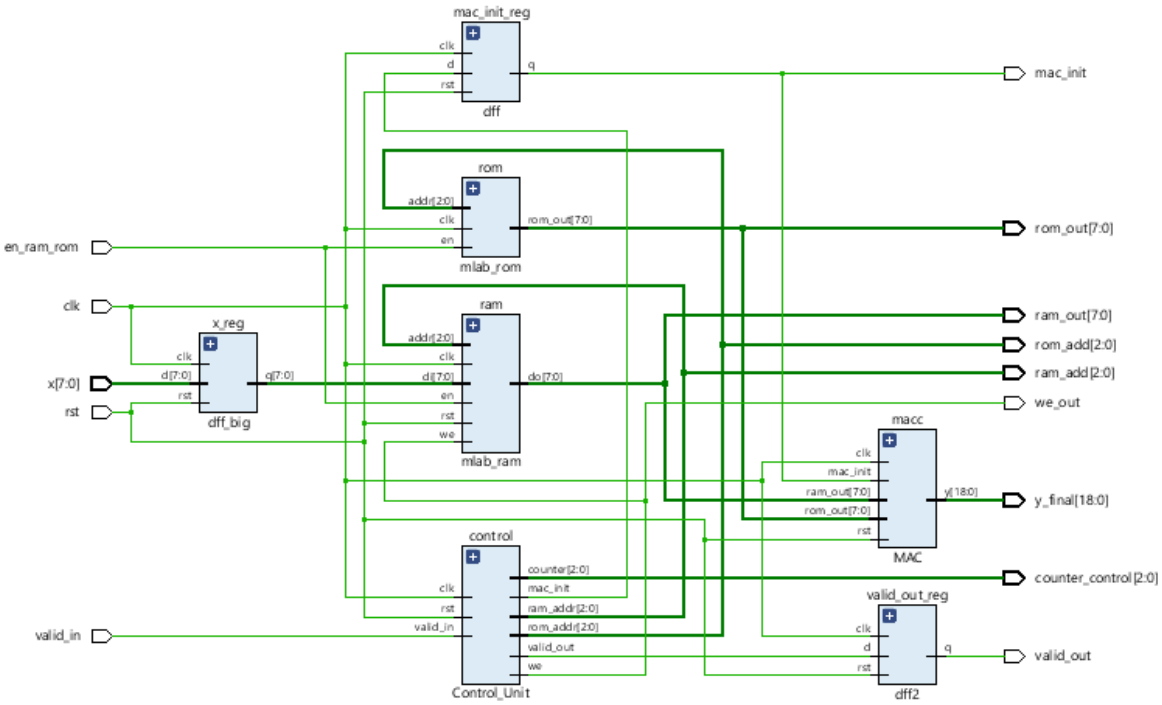
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity FIR is
  Port ( clock : in std_logic;
         reset : 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;
         fir_output : out std_logic_vector (18 downto 0);
         rom_output,ram_output:out STD_LOGIC_VECTOR (7 downto 0);
         rom_address,ram_address:out STD_LOGIC_VECTOR (2 downto 0);
         mac_init : out std_logic;
         we_out:out std_logic
         );
end FIR;

architecture Behavioral of FIR is
    component mlab_rom is
       generic (
            coeff_width : integer :=8
       );
    Port ( clock : in  STD_LOGIC;
              en : in  STD_LOGIC;
           addr : in  STD_LOGIC_VECTOR (2 downto 0);
           rom_output : out  STD_LOGIC_VECTOR (coeff_width-1 downto
0));
    end component;

    component mlab_ram is
         generic (
            data_width : integer :=8
         );
        port (clock  : in std_logic;
              reset  : in std_logic;
              we     : in std_logic;
              en     : in std_logic;
```

```
          addr : in std_logic_vector(2 downto 0);
          di   : in std_logic_vector(data_width-1 downto 0);
          do   : out std_logic_vector(data_width-1 downto 0));
end component;

component Control_Unit is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           valid_in : in STD_logic;
           mac_init: out STD_LOGIC;
           ram_address : out STD_LOGIC_VECTOR (2 downto 0);
           rom_address : out STD_LOGIC_VECTOR (2 downto 0);
           we: out STD_LOGIC;
           valid_out: out STD_LOGIC
           );
end component;

component MAC is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           rom_output : in STD_LOGIC_VECTOR (7 downto 0);
           ram_output : in STD_LOGIC_VECTOR (7 downto 0);
           mac_init : in STD_LOGIC;
           y : out STD_LOGIC_VECTOR (18 downto 0));
end component;

component dff is
 port(
      d : in  std_logic;
      q : out  std_logic;
      clock : in std_logic;
      reset : in std_logic
     );
end component;

component dff_8bit is
    Port ( d : in STD_LOGIC_VECTOR (7 downto 0);
           clock : in STD_LOGIC;
           reset : in STD_LOGIC;
```

```
            q : out STD_LOGIC_VECTOR (7 downto 0));
    end component;

    component dff2 is
        port(
            d : in  std_logic;
            q : out  std_logic;
            clock : in std_logic;
            reset : in std_logic
           );
    end component;

    signal x_dff,ram_output_temp,rom_output_temp : std_logic_vector(7
downto 0);
    signal ram_address_temp, rom_address_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_8bit port map(d=>x, clock=>clock, reset=>reset,
q=>x_dff);
    control: Control_Unit port map(clock=>clock, reset=>reset,
valid_in=>valid_in, mac_init=>mac_init_temp,
ram_address=>ram_address_temp,
rom_address=>rom_address_temp,we=>we_temp,valid_out=>valid_out_temp);
    mac_init_reg : dff port map(d=>mac_init_temp, clock=>clock,
reset=>reset, q=>mac_init_dff);
    valid_out_reg : dff2 port map(d=>valid_out_temp, clock=>clock,
reset=>reset, q=>valid_out_dff);
    rom: mlab_rom port map(clock=>clock,en=>en_ram_rom,
addr=>rom_address_temp,rom_output=>rom_output_temp);
    ram: mlab_ram port map(clock=>clock, reset=>reset,
en=>en_ram_rom, we=>we_temp ,addr=>ram_address_temp,di=>x_dff,
do=>ram_output_temp);
    macc: MAC port map(reset=>reset,clock=>clock,
rom_output=>rom_output_temp, ram_output=>ram_output_temp,
mac_init=>mac_init_dff, y=>fir_output);
```

```
mac_init <=mac_init_dff;
we_out <= we_temp;
ram_output <=ram_output_temp;
rom_output <=rom_output_temp;
rom_address <=rom_address_temp;
ram_address <=ram_address_temp;
valid_out <=valid_out_dff;
end Behavioral;
```

Το RTL Schematic που προκύπτει φαίνεται παρακάτω:



Για την υλοποίηση του testbench χρησιμοποιήθηκε ο παρακάτω κώδικας:

```
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 Behavioral of FIR_tb is

    constant CLK_PERIOD : time := 10 ns;
    constant CLK_PERIOD_ALL: time:=8*CLK_PERIOD;

    signal clk : std_logic := '0';
    signal rst : std_logic := '1';
    signal en_ram_rom : std_logic := '0';
    signal x : std_logic_vector(7 downto 0) := (others => '0');
    signal y_final : std_logic_vector(18 downto 0);
    signal rom_out, ram_out: std_logic_vector(7 downto 0);
    signal rom_add, ram_add, counter_control : std_logic_vector(2
downto 0);
    signal mac_init : std_logic;
    signal valid_out : std_logic;
    signal we_out : std_logic := '1';
    signal valid_in: std_logic := '0';

begin

    DUT: entity work.FIR
    Port map (
        clk => clk,
        rst => rst,
        en_ram_rom => en_ram_rom,
        x => x,
        y_final => y_final,
        rom_out => rom_out,
        ram_out => ram_out,
        rom_add => rom_add,
        ram_add => ram_add,
        we_out => we_out,
        mac_init => mac_init,
        valid_in => valid_in,
        valid_out => valid_out,
        counter_control => counter_control
     -- RAM_IN => RAM_IN
    );
```

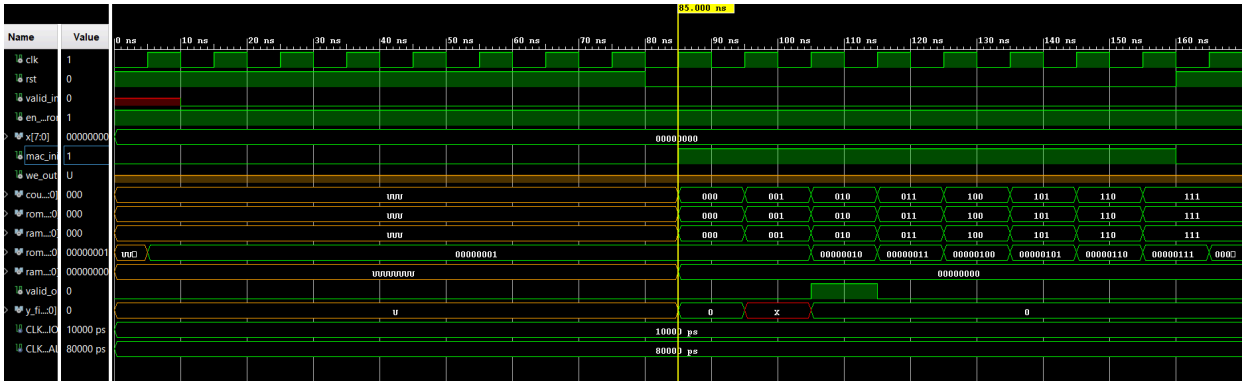
```
clk_process : process
begin
    while now < 10000 ns loop
        clk <= '0';
        wait for CLK_PERIOD/2;
        clk <= '1';
        wait for CLK_PERIOD/2;
    end loop;
    wait;
end process clk_process;

validin_process : process
begin
    while now < 10000 ns loop
        valid_in <= '1';
        wait for 1*CLK_PERIOD;
        valid_in <= '0';
        wait for 20*CLK_PERIOD;
    end loop;
end process validin_process;


stim_process : process
begin
    rst <= '0';
    en_ram_rom <= '1';

    rst <= '1';
    valid_in<='0';
    x <= "00000000";
    wait for CLK_PERIOD_ALL;
    rst <= '0';
    valid_in<='0';
    x <= "00000000";
    wait for CLK_PERIOD_ALL;
    rst <= '1';
    valid_in<='0';
    x <= "00000000";
    wait for CLK_PERIOD_ALL;
```

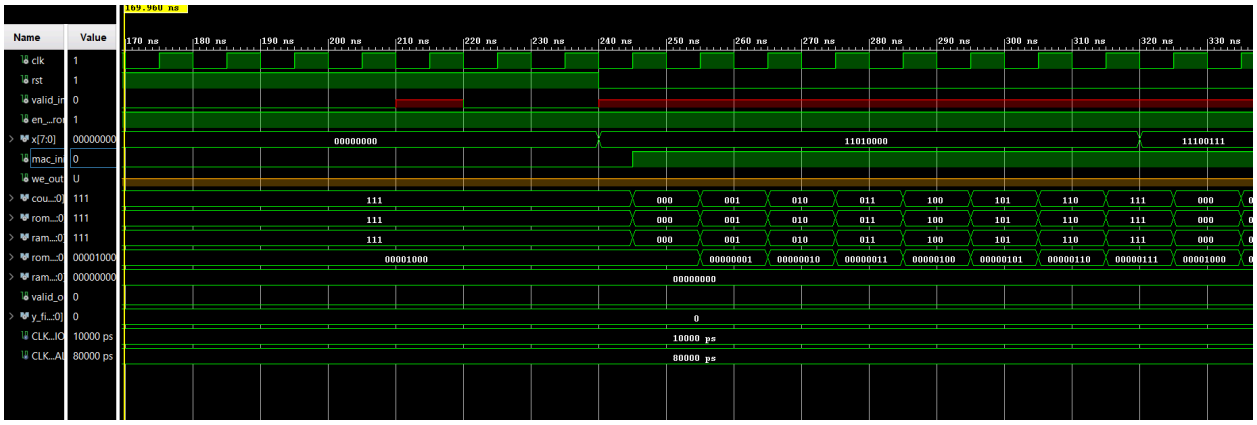
```
rst <= '0';
valid_in<='1';
x <= "11010000";
wait for CLK_PERIOD_ALL ;
rst <= '0';
valid_in<='1';
x <= "11100111";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "00100000";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11101001";
wait for CLK_PERIOD_ALL ;
rst <= '0';
valid_in<='1';
x <= "10100001";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "00011000";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "01000111";
wait for CLK_PERIOD_ALL ;
rst <= '0';
valid_in<='1';
x <= "10001100";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11110101";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11110111";
```

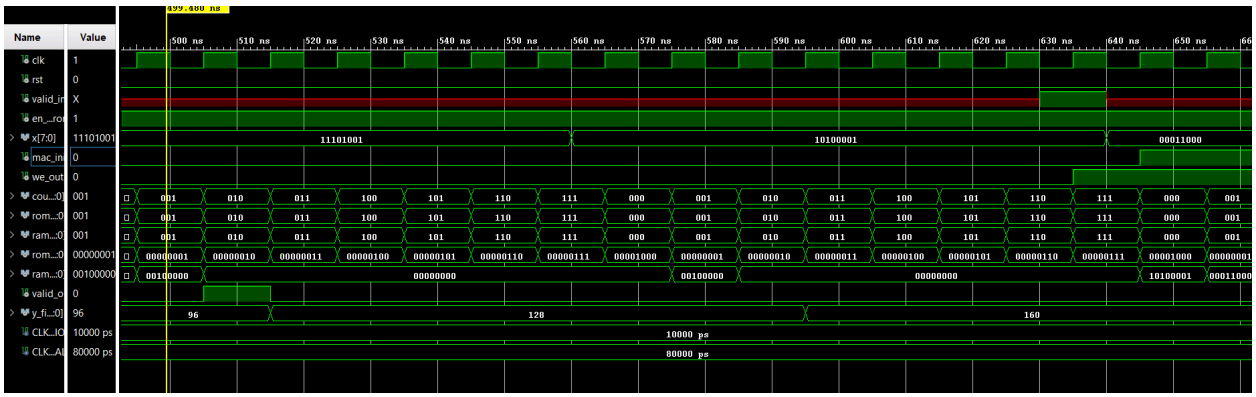
```
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "00101000";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11111000";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11110101";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "01111100";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11001100";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "00100100";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "01101011";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11101010";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
x <= "11001010";
wait for CLK_PERIOD_ALL;
rst <= '0';
valid_in<='1';
```

```
        x <= "11110101";
        wait for CLK_PERIOD_ALL;
        rst <= '0';
        valid_in<='1';
        x <= "00000000";
        wait for CLK_PERIOD_ALL;

        wait;
    end process stim_process;
end Behavioral;
```

Το simulation που προκύπτει με βάσει τα παραπάνω είναι το κάτωθι:







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 1η περίπτωση, περιμένει να ολοκληρωθεί ο προηγούμενος υπολογισμός και ορθώς δεν δέχεται την ενδιάμεση είσοδο (τα mac_init, we_out παραμένουν στο 0). Στη 2η περίπτωση, παρατηρούμε ότι, όσο δεν υπάρχει νέα τιμή εισόδου, το ram_out παραμένει σταθερό, ενώ όταν λαμβάνεται εκ νλεου τιμή valid_in = '1', τότε γίνεται και πάλι υπολογισμός από το φίλτρο και η έξοδος φαίνεται μετά από 8 κύκλους ρολογιού.

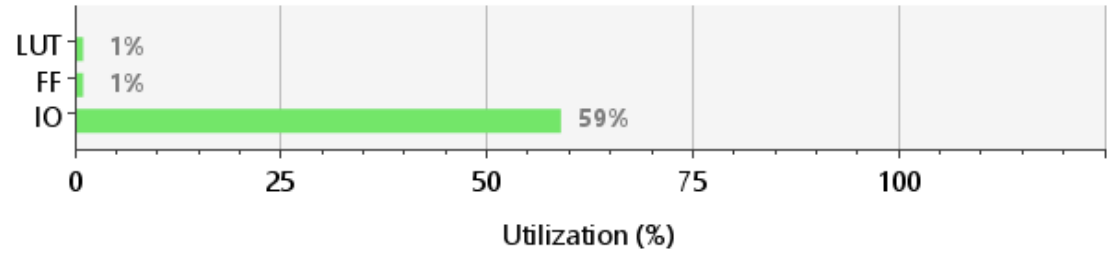
Για το critical path και το utilization report:

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
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	∞	
Path 4	∞	9	9	20	rom/rom_out_reg[1]/C	macc/sum_reg[13]/D	5.503	2.589	2.914	∞	
Path 5	∞	9	9	20	rom/rom_out_reg[1]/C	macc/sum_reg[15]/D	5.484	2.570	2.914	∞	
Path 6	∞	9	9	20	rom/rom_out_reg[1]/C	macc/sum_reg[14]/D	5.411	2.497	2.914	∞	
Path 7	∞	9	9	20	rom/rom_out_reg[1]/C	macc/sum_reg[12]/D	5.390	2.476	2.914	∞	
Path 8	∞	7	7	9	ram/do_reg[3]/C	macc/sum_reg[9]/D	5.293	2.548	2.745	∞	
Path 9	∞	7	7	9	ram/do_reg[3]/C	macc/sum_reg[11]/D	5.274	2.529	2.745	∞	
Path 10	∞	7	7	9	ram/do_reg[3]/C	macc/sum_reg[10]/D	5.201	2.456	2.745	∞	

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack:                    inf
  Source:                 rom/rom_out_reg[1]/C
                            (rising edge-triggered cell FDRE)
  Destination:            macc/sum_reg[17]/D
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        5.617ns  (logic 2.703ns (48.122%)  route 2.914ns (51.878%))
  Logic Levels:           10  (CARRY4=5 FDRE=1 LUT2=1 LUT4=1 LUT5=1 LUT6=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDRE                         0.000     0.000 r  rom/rom_out_reg[1]/C
                         FDRE (Prop_fdre_C_Q)         0.456     0.456 r  rom/rom_out_reg[1]/Q
                         net (fo=20, unplaced)        1.025     1.481    rom/Q[1]


                         LUT4 (Prop_lut4_I1_O)        0.120     2.356 r  ram/multOp__0_carry__0_i_1/O
                         net (fo=2, unplaced)         0.500     2.856    rom/multOp__0_carry__0_1[0]
                         LUT5 (Prop_lut5_I4_O)        0.327     3.183 r  rom/multOp__0_carry__0_i_5/O
                         net (fo=1, unplaced)         0.000     3.183    macc/sum[4]_i_9_1[3]
                         CARRY4 (Prop_carry4_S[3]_CO[3])
                                                      0.401     3.584 r  macc/multOp__0_carry__0/CO[3]
                         net (fo=1, unplaced)         0.000     3.584    macc/multOp__0_carry__0_n_0
                         CARRY4 (Prop_carry4_CI_CO[3])
                                                      0.114     3.698 r  macc/multOp__0_carry__1/CO[3]
                         net (fo=2, unplaced)         0.929     4.627    rom/CO[0]
                         LUT2 (Prop_lut2_I0_O)        0.117     4.744 r  rom/sum[8]_i_2/O
                         net (fo=1, unplaced)         0.000     4.744    macc/sum_reg[11]_0[0]
                         CARRY4 (Prop_carry4_DI[3]_CO[3])
                                                      0.411     5.155 r  macc/sum_reg[8]_i_1/CO[3]
                         net (fo=1, unplaced)         0.000     5.155    macc/sum_reg[8]_i_1_n_0
                         CARRY4 (Prop_carry4_CI_CO[3])

                                                      0.348     5.617 r  macc/sum_reg[16]_i_1/O[1]
                         net (fo=1, unplaced)         0.000     5.617    macc/sum_reg[16]_i_1_n_6
                         FDCE                                        r  macc/sum_reg[17]/D
  -------------------------------------------------------------------    -------------------
```

```
Min Delay Paths
--------------------------------------------------------------------------------
Slack:                    inf
  Source:                 control/valid_out_reg/C
                            (rising edge-triggered cell FDCE)
  Destination:            valid_out_reg/delay3/q_reg/D
  Path Group:             (none)
  Path Type:              Min at Fast Process Corner
  Data Path Delay:        0.282ns  (logic 0.141ns (50.038%)  route 0.141ns (49.962%))
  Logic Levels:           1  (FDCE=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDCE                         0.000     0.000 r  control/valid_out_reg/C
                         FDCE (Prop_fdce_C_Q)         0.141     0.141 r  control/valid_out_reg/Q
                         net (fo=1, unplaced)         0.141     0.282    valid_out_reg/delay3/d
                         FDCE                                         r  valid_out_reg/delay3/q_reg/D
```

Resource	Utilization	Available	Utilization %
LUT	86	17600	0.49
FF	135	35200	0.38
IO	59	100	59.00



Σύμφωνα με τα παραπάνω, το critical path του φίλτρου είναι αυτό που ξεκινά από το rom_out και καταλήγει στο mac, με συνολική καθυστέρηση ίση με 5.617 ns. Αν και η καθυστέρηση είναι πολύ μικρή, αξίζει να σημειωθεί πως αξιοποιείται μεγάλος αριθμός πόρων. Αυτό μπορεί να οφείλεται στο γεγονός ότι τα components του FIR χρησιμοποιούν το ρολόι εισόδου, το οποίο θα μπορούσε να αποφευχθεί, ιδιαίτερα στην περίπτωση του mac_unit, η αρχικοποίηση του οποίου γίνεται μέσω του σήματος mac_init του control_unit. ,