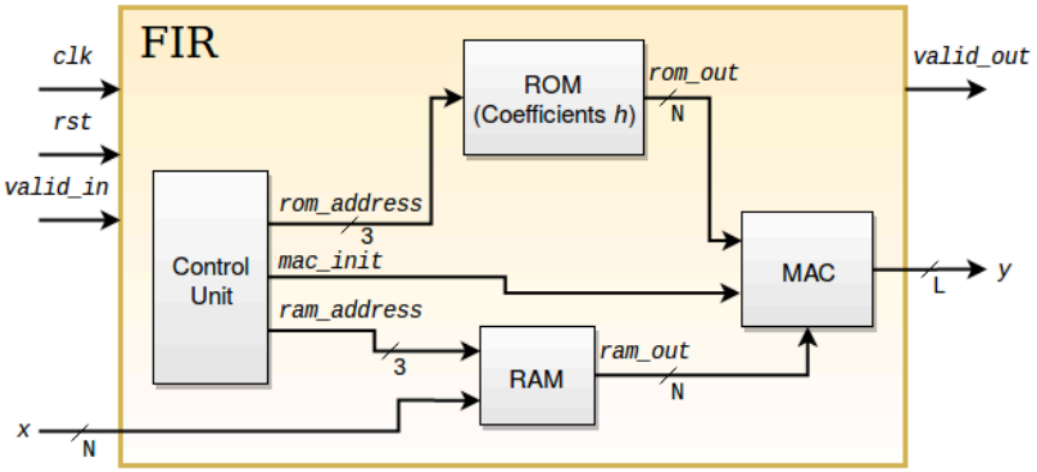
Εθνικό Μετσόβιο Πολυτεχνείο, ΣΗΜΜΥ
Ψηφιακά Συστήματα VLSI

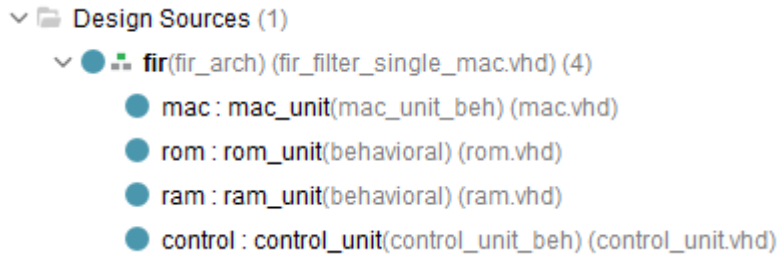
4η Εργαστηριακή Άσκηση:
Υλοποίηση FIR Φίλτρου

Ημ/νια : 27.04.2024
Ομάδα 19
Θεοδώρα Εξάρχου, ΑΜ : 03120865
Παναγιώτης Μπέλσης, ΑΜ: 03120874

1) Να υλοποιήσετε το παραπάνω φίλτρο λαμβάνοντας υπόψη όλες τις ζητούμενες λειτουργίες του.



Ιεράρχηση fir design sources



Υλοποιησαμε το παραπάνω φιλτρο ως εξης :

Κώδικες παραπάνω αρχείων:

FIR Filter

● Περιλαμβάνει ασύγχρονο reset το οποίο αρχικοποιεί τη μονάδα Control Unit, τη μνήμη RAM με ‘0’ καθώς και το σήμα εγκυρότητας εξόδου valid_out.

fir_filter_single_mac.vhd

```
----------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity fir is

 port (
 clk,valid_in,rst: in std_logic;
 x: in std_logic_vector (7 downto 0);
 y: out std_logic_vector (18 downto 0);
 valid_out: out std_logic
 );
end fir;
architecture fir_arch of fir is
 component mac_unit is
 port (
 clk,mac_init,en: in std_logic;
 rom_out, ram_out: in std_logic_vector(7 downto 0);
 result: out std_logic_vector(18 downto 0)
 );
 end component;

 component rom_unit is
 port (
 clk,en : in STD_LOGIC;
 addr : in STD_LOGIC_VECTOR (2 downto 0);
 rom_out : out STD_LOGIC_VECTOR (7 downto 0)
 );

 end component;
 component ram_unit is
 port (
 clk,we,en,rst: in std_logic;
 addr : in std_logic_vector(2 downto 0);
 di : in std_logic_vector(7 downto 0);
 do : out std_logic_vector(7 downto 0)
 );
```

```
 end component;
 component control_unit is
 port (
 clk , valid_in , rst: in std_logic;
 rom_addr  , ram_addr: out std_logic_vector(2 downto 0);
 mac_init  , we , en , valid_out: out std_logic
 );
 end component;

 signal reg_mac_init, mac_init_signal, en_signal, we_signal: std_logic;
 signal reg_x, rom_out_signal,ram_out_signal: std_logic_vector(7 downto
0);
 signal rom_addr_signal,ram_addr_signal: std_logic_vector(2 downto 0);
 begin
 mac: mac_unit port map(clk=>clk,en=>en_signal, mac_init=>reg_mac_init,
rom_out=>rom_out_signal, ram_out=>ram_out_signal, result=>y);
 rom: rom_unit port map(clk=>clk, en=>en_signal,
addr=>rom_addr_signal, rom_out=>rom_out_signal);
 ram: ram_unit port map(clk=>clk, we=>we_signal, en=>en_signal,
rst=>rst, addr=>ram_addr_signal, di=>reg_x, do=>ram_out_signal);
 control: control_unit port map(clk=>clk, valid_in=>valid_in,
rst=>rst, rom_addr=>rom_addr_signal, ram_addr=>ram_addr_signal,
mac_init=>mac_init_signal, we=>we_signal, en=>en_signal,
valid_out=>valid_out);

 process(clk)
 begin
 if(rising_edge(clk)) then
 reg_mac_init <= mac_init_signal;
 reg_x <= x;

 end if;
 end process;

end fir_arch;
--------------------------------------------------------------------------
```

Control Unit - Μονάδα Ελέγχου:

●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.
● Δέχεται μία νέα μη προσημασμένη (unsigned) τιμή εισόδου x κάθε 8 κύκλους ρολογιού.
●Ανα 8 κυκλους δινει μεσω του σηματος **we=1** εντολή να γράψει τις νεες τιμες του x που

εισερχονται στη Ram
● Για κάθε νέα είσοδο x δέχεται ένα σήμα εγκυρότητας εισόδου (valid_in) το οποίο υποδηλώνει την εγκυρότητα της συγκεκριμένης εισόδου.
●Περιλαμβάνει ασύγχρονο reset το οποίο αρχικοποιεί τη μονάδα

control_unit.vhd

```
----------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity control_unit is

    port (
       clk, valid_in, rst: in std_logic;
       rom_addr, ram_addr: out std_logic_vector(2 downto 0);
       mac_init, we, en, valid_out: out std_logic
    );

end control_unit;

architecture control_unit_beh of control_unit is

    signal reg_valid_out: std_logic_vector(8 downto 0) :=
(others=>'0');
    signal counter: std_logic_vector(2 downto 0) := "000";

    begin

        process(clk,rst)
        begin
            if (rst='1') then

                counter <= "000";
                ram_addr <= "000";
                rom_addr <= "000";
                reg_valid_out <= (others => '0');


            elsif (rising_edge(clk)) then

                    if (counter="000") and (valid_in='1')then

                        mac_init <= '1';
                        we <= '1';
```

```
                        en <= '1';
                       reg_valid_out(0) <= '1';
                        valid_out <= reg_valid_out(8);
                       reg_valid_out(8 downto 1) <= reg_valid_out(7 downto
0);
                    elsif (counter="000") and (valid_in='0')then
                        mac_init <= '1';
                        en <= '0';
                        we <= '0';
                      reg_valid_out(0) <= '0';
                      valid_out <= reg_valid_out(8);
                      reg_valid_out(8 downto 1) <= reg_valid_out(7 downto
0);
                   elsif counter="001" then
                      en <= '1';
                      we <= '0';
                      mac_init <= '0';
                      reg_valid_out(0) <= '0';
                      valid_out <= reg_valid_out(8);
                      reg_valid_out(8 downto 1) <= reg_valid_out(7 downto
0);
                   else
                      en <= '1';
                      we <= '0';
                      mac_init <= '0';
                      valid_out <= reg_valid_out(8);
                      reg_valid_out(8 downto 1) <= reg_valid_out(7 downto
0);

                    end if;
            if counter/=7 then
                counter<=counter+1;
            else
               counter<=(others=>'0');
            end if;
              ram_addr <= counter;
              rom_addr <= counter;
            end if;

      end process;
end control_unit_beh;
-- ----------------------------------------------------------------------
```

RAM

●Αποθηκεύει την παρούσα τιμή του σήματος εισόδου x, καθώς και τις 7 προηγούμενες, που είναι απαραίτητες για τον υπολογισμό της εξόδου y.
● Δέχεται ως είσοδο μία διεύθυνση (ram_address) και δίνει στην έξοδο την αντίστοιχη τιμή του σήματος εισόδου που είναι αποθηκευμένη σε αυτή (ram_out).

ram.vhd

```
-- ----------------------------------------------------------------------------

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

entity ram_unit is

  generic (
  data_width : integer :=8);

    port (
          clk,we,en,rst:  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 ram_unit;

    architecture behavioral of ram_unit is

    type ram_type is array (7 downto 0) of std_logic_vector
(data_width-1 downto 0);

begin
    process (clk,rst)
        variable RAM : ram_type := (others => (others => '0'));
    begin
        if (rst='1') then
            RAM := (others=> (others=>'0'));
        elsif (rising_edge(clk)) then
            if (en = '1') then
                if (we = '1') then
                    RAM(7 downto 1) := RAM(6 downto 0);
                    RAM(0) := di;
                    do <= di;
                else
                    do <= RAM(conv_integer(addr));
                end if;
            end if;
        end if;
    end process;
```

```
 end behavioral;
```

-- --

ROM:

●Έχει αποθηκευμένους του 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) σας παρέχεται έτοιμη και πρέπει να την ενσωματώσετε κατάλληλα στη συνολική αρχιτεκτονική του φίλτρου.

rom.vhd

-- --

```
 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use ieee.std_logic_unsigned.all;
 entity rom_unit is
 generic (
 coeff_width : integer :=8
 );
  port (
  clk : in STD_LOGIC;
  en : in STD_LOGIC;
  addr : in STD_LOGIC_VECTOR (2 downto 0);
  rom_out : out STD_LOGIC_VECTOR (coeff_width-1 downto 0)
  );
 end rom_unit;
 architecture behavioral of rom_unit 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 (rising_edge(clk)) then
  if (en = '1') then
  rom_out <= rdata;
  end if;
  end if;
```

```
  end process;
end behavioral;
- ---------------------------------------------------------------------------
```

MAC (Multiplier Accumulator Unit)

●Δέχεται ως είσοδο τους 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.
● Για κάθε έγκυρη τιμή της εξόδου y ενεργοποιείται ένα σήμα εγκυρότητας εξόδου (valid_out).

mac.vhd

```
-----------------------------------------------------------------------------------------------------------------------
```

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


entity mac_unit is

    port (
        clk,mac_init: in std_logic;
        rom_out, ram_out: in std_logic_vector(7 downto 0);
        result: out std_logic_vector(18 downto 0)

    );

end mac_unit;

architecture mac_unit_beh of mac_unit is

    begin
        process(clk)

        variable acc: std_logic_vector(18 downto 0) := (others=>'0');
        begin

            if (rising_edge(clk)) then
                if (mac_init='1') then
                    acc := (others=>'0');
                    acc(15 downto 0) := rom_out * ram_out;
                else
```

```
                    acc := acc + rom_out * ram_out;
                end if;
                result <= acc;

            end if;
        end process;

end mac_unit_beh;
—------------------------------------------------------------------------
```

2) Να δημιουργήσετε testbench για τον έλεγχο της ορθής λειτουργίας του φίλτρου για N=8. Το testbench να περιλαμβάνει τον έλεγχο όλων των λειτουργιών του φίλτρου.

Στο testbench του FIR βάλαμε ως δεδομένα εισόδου x:
208 231 32 233 161 24 71 140 245 247 40 248 245 124 204 36 107 234 202 245

Τους αριθμούς τους εισάγουμε ανά 8 κύκλους όπως μας ζητήθηκε ,και ανά 400ns κάνουμε Reset ,ενώ όλες οι τιμές είναι αρχικοποιημένες με Valid In=1 .

—----------

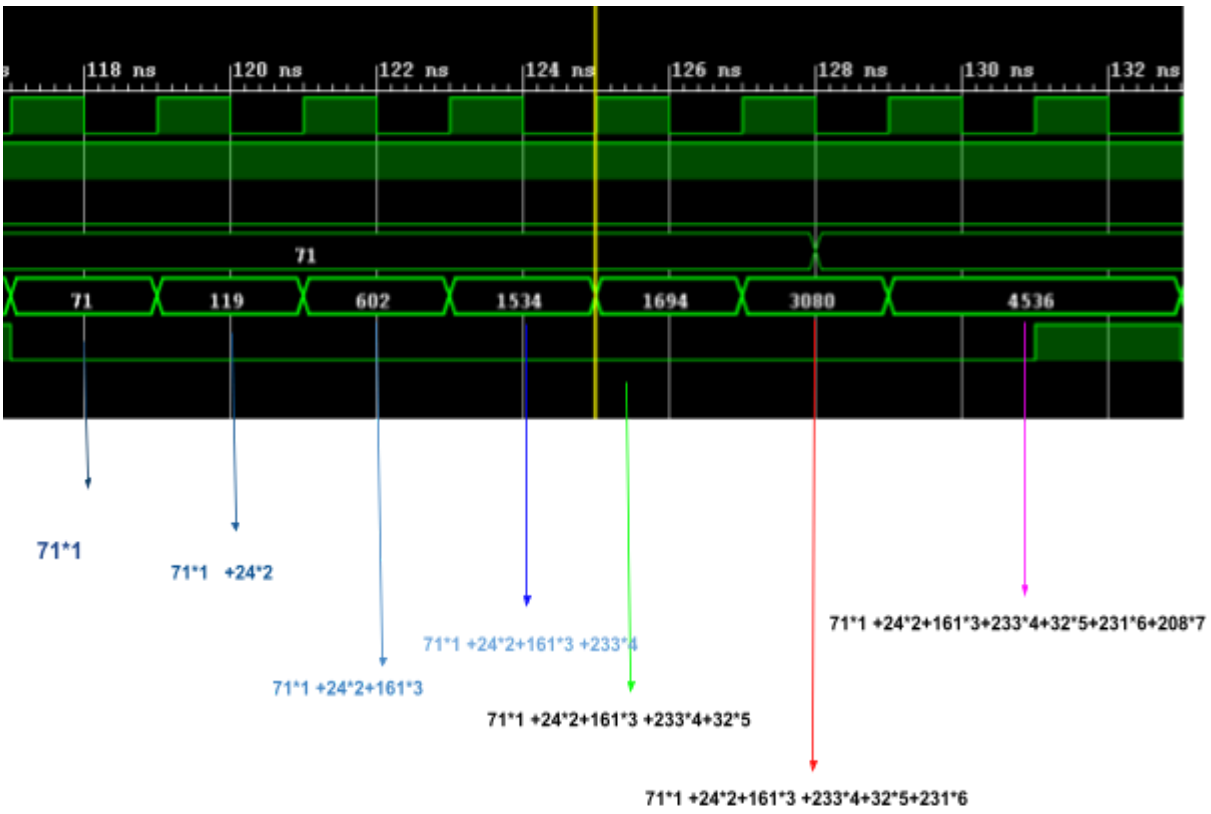
```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY fir_tb IS
END fir_tb;

ARCHITECTURE fir_tb_arch OF fir_tb IS
    SIGNAL clk, valid_in, rst: std_logic := '0';
    SIGNAL x: std_logic_vector(7 downto 0) := (others=>'0');
    SIGNAL y: std_logic_vector(18 downto 0) := (others=>'0');
    SIGNAL valid_out: std_logic;

    COMPONENT fir IS
        PORT (
            clk, valid_in, rst: in std_logic;
            x: in std_logic_vector(7 downto 0);
            y: out std_logic_vector(18 downto 0);
            valid_out: out std_logic
        );
    END COMPONENT;

BEGIN
    DUT: fir PORT MAP(
```

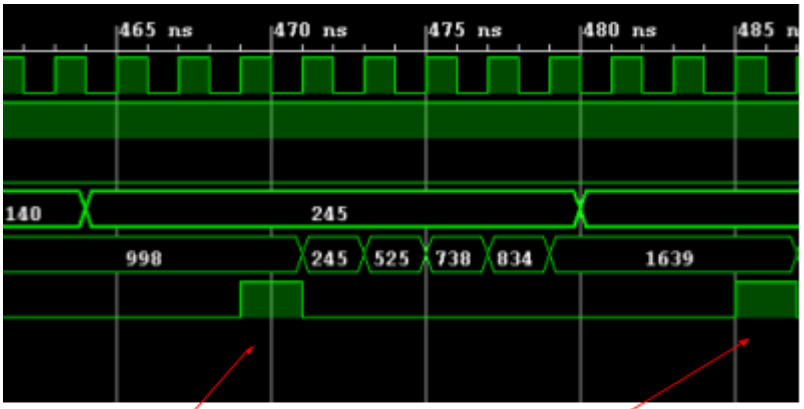
```
    clk => clk,
    valid_in => valid_in,
    rst => rst,
    x => x,
    y => y,
    valid_out => valid_out
);

CHANGE_RST: PROCESS
begin
    rst <= '0';
    wait for 400 ns;
    rst <= '1';
    wait for 2 ns;
END PROCESS;

CLOCK: PROCESS
begin
    clk <= '0';
    wait for 1 ns;
    clk <= '1';
    wait for 1 ns;
END PROCESS;

CHANGE_X: PROCESS
begin
    wait for 16 ns;
    x <= "11010000";
    wait for 16 ns;
    x <= "11100111";
    wait for 16 ns;
    x <= "00100000";
    wait for 16 ns;
    x <= "11101001";
    wait for 16 ns;
    x <= "10100001";
    wait for 16 ns;
    x <= "00011000";
    wait for 16 ns;
    x <= "01000111";
    wait for 16 ns;
    x <= "10001100";
    wait for 16 ns;
    x <= "11110101";
    wait for 16 ns;
    x <= "11110111";
```

```
        wait for 16 ns;
        x <= "00101000";
        wait for 16 ns;
        x <= "11111000";
        wait for 16 ns;
        x <= "11110101";
        wait for 16 ns;
        x <= "01111100";
        wait for 16 ns;
        x <= "11001100";
        wait for 16 ns;
        x <= "00100100";
        wait for 16 ns;
        x <= "01101011";
        wait for 16 ns;
        x <= "11101010";
        wait for 16 ns;
        x <= "11001010";
        wait for 16 ns;
        x <= "11110101";
    END PROCESS;

    CHANGE_VIN: PROCESS
    begin
        valid_in <= '1';
        wait for 1000 ns;

    END PROCESS;

END fir_tb_arch;
```

Με την παρακάτω φωτογραφία από το simulation μας επιβεβαιώνουμε οτι λειτουργεί η συνάρτηση που υλοποιεί το FIR φίλτρο.

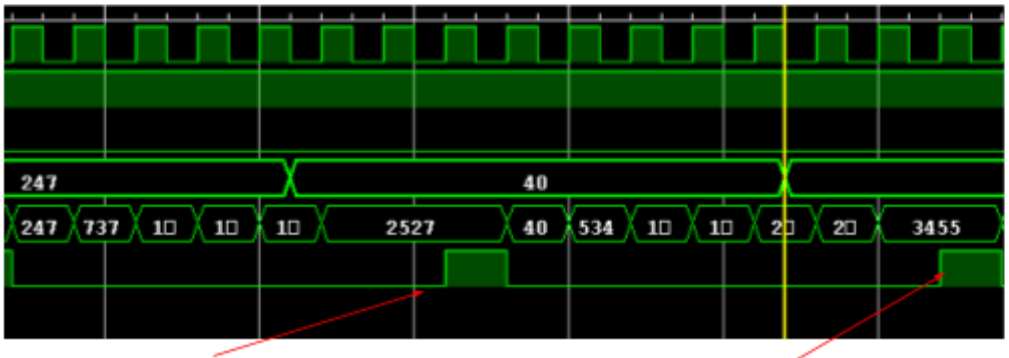
$$y[n] = \sum_{k=0}^{M} h[k]\,x[n-k] = h[0]\,x[n] \;+\; h[1]\,x[n-1] \;+\; \ldots \;+\; h[M]\,x[n-M] \qquad (1)$$



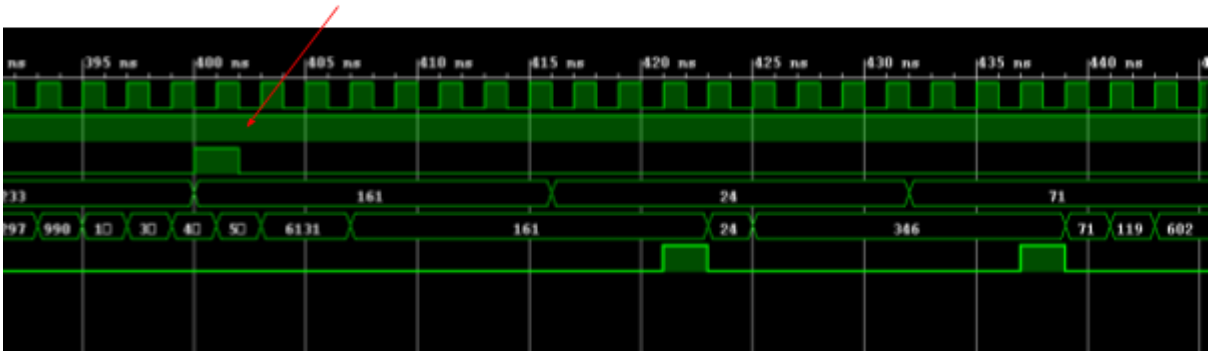


Με τα κοκκινα βελακια παρατηρουμε τις στιγμες που εουμε valid out τιμη στο simulation μας

συνεχεια simulation

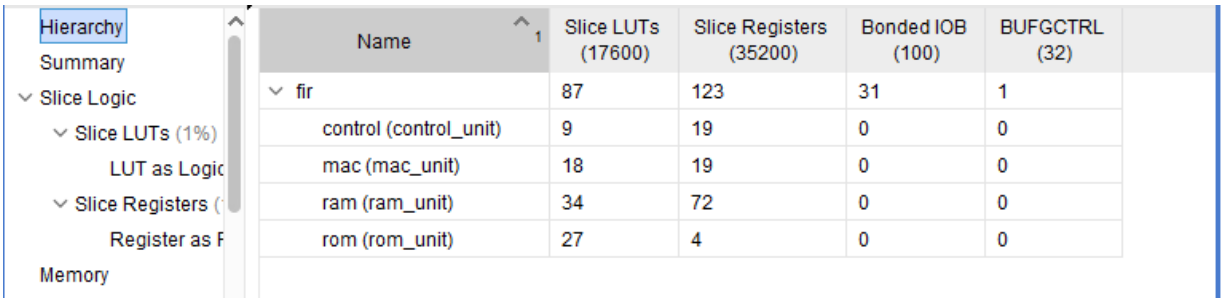


Παρατηρουμε την στιγμη που το κυκλωμα μας κανει reset, και με αυτο τον τροπο επαληθευουμε οτι μετα το reset ,γινεται αρχικοποιηση που μας ζητηθηκε στις μοναδες

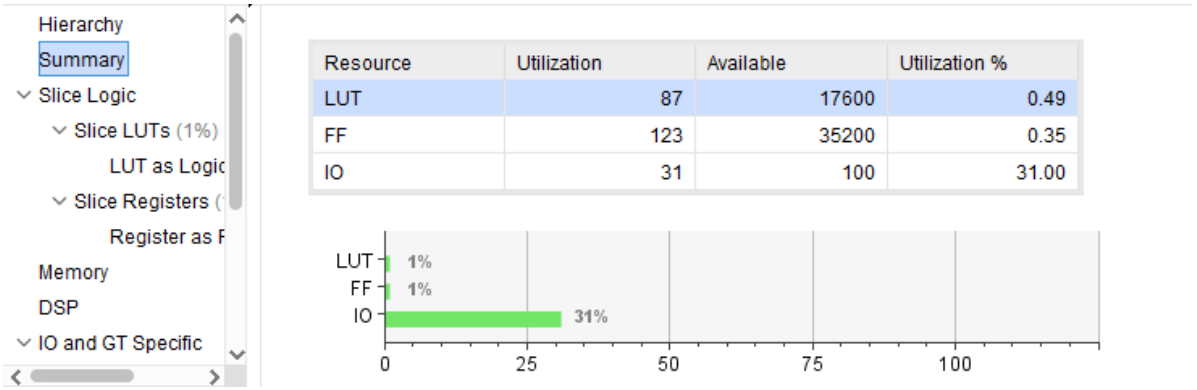


3) Να καταγράψετε τους πόρους (resources) που χρησιμοποιήθηκαν για την υλοποίηση του φίλτρου καθώς και τη συχνότητα λειτουργίας του FPGA ελέγχοντας το κρίσιμο μονοπάτι.

Τα resources του FPGA που χρησιμοποιήθηκαν για την υλοποίηση του fir φίλτρου φαίνονται στα παρακάτω screenshots.



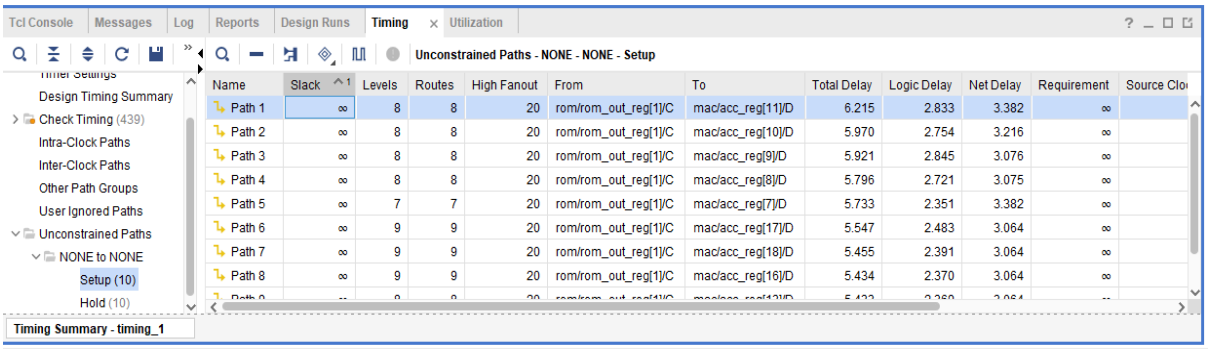
Name	Slice LUTs (17600)	Slice Registers (35200)	Bonded IOB (100)	BUFGCTRL (32)
fir	87	123	31	1
control (control_unit)	9	19	0	0
mac (mac_unit)	18	19	0	0
ram (ram_unit)	34	72	0	0
rom (rom_unit)	27	4	0	0



Resource	Utilization	Available	Utilization %
LUT	87	17600	0.49
FF	123	35200	0.35
IO	31	100	31.00

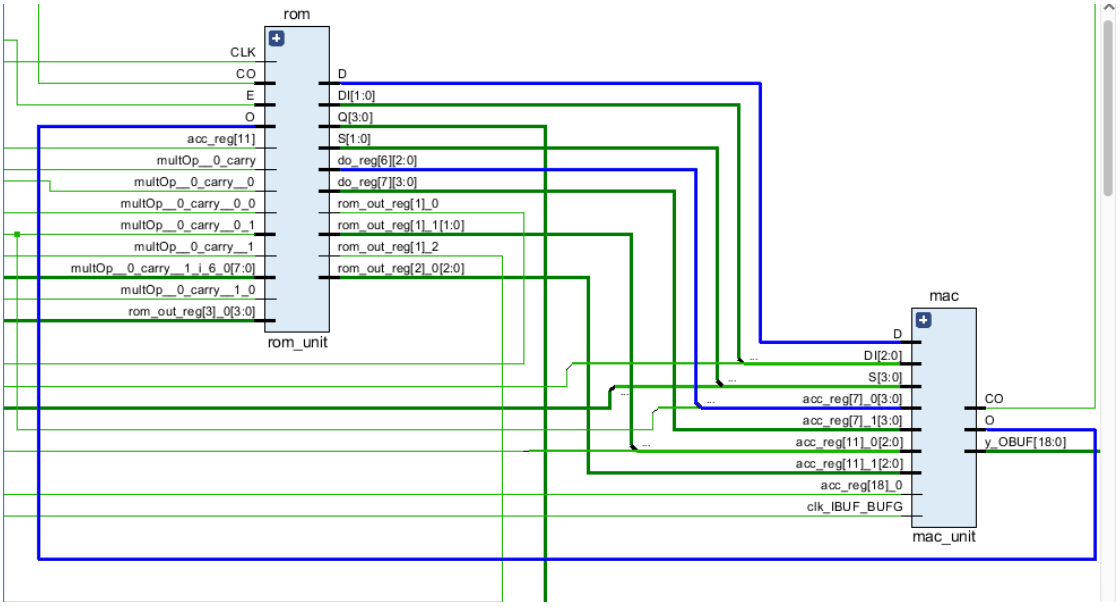
Εύρεση συχνότητας λειτουργίας του FPGA.

Το κρίσιμο μονοπάτι είναι το παρακάτω Path1 με **total delay = 6.215ns**



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clo
Path 1	∞	8	8	20	rom/rom_out_reg[1]/C	mac/acc_reg[11]/D	6.215	2.833	3.382	∞	
Path 2	∞	8	8	20	rom/rom_out_reg[1]/C	mac/acc_reg[10]/D	5.970	2.754	3.216	∞	
Path 3	∞	8	8	20	rom/rom_out_reg[1]/C	mac/acc_reg[9]/D	5.921	2.845	3.076	∞	
Path 4	∞	8	8	20	rom/rom_out_reg[1]/C	mac/acc_reg[8]/D	5.796	2.721	3.075	∞	
Path 5	∞	7	7	20	rom/rom_out_reg[1]/C	mac/acc_reg[7]/D	5.733	2.351	3.382	∞	
Path 6	∞	9	9	20	rom/rom_out_reg[1]/C	mac/acc_reg[17]/D	5.547	2.483	3.064	∞	
Path 7	∞	9	9	20	rom/rom_out_reg[1]/C	mac/acc_reg[18]/D	5.455	2.391	3.064	∞	
Path 8	∞	9	9	20	rom/rom_out_reg[1]/C	mac/acc_reg[16]/D	5.434	2.370	3.064	∞	

Κρίσιμο Μονοπάτι όπως φαίνεται στο Schematic



Συνεπώς η μέγιστη συχνότητα λειτουργίας του FPGA είναι : **fmax = 160.9 MHz .**