Ψηφιακά Συστήματα VLSI

8ο εξάμηνο, Ακαδημαϊκή περίοδος 2024-2025

4η Εργαστηριακή Άσκηση

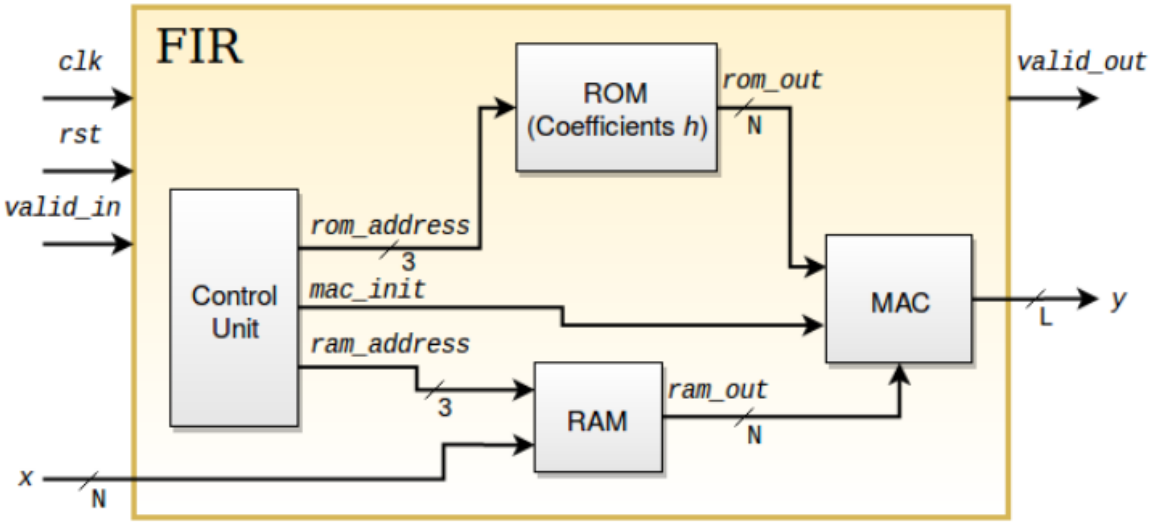
Δημήτρης Καμπανάκης: 03121012
Αγγελική Ζέρβα: 03121101

Εισαγωγή

Ένα fir φίλτρο έχει την γενική μορφή:

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

όπου:
M: τάξη του φίλτρου y[n]: η διακριτή έξοδος του φίλτρου την χρονική στιγμή n h[k]: ο k-οστός συντελεστής του φίλτρου x[n]: η τιμή του σήματος εισόδου την χρονική στιγμή n
Στόχος μας σε υτήν την εργαστηριακή άσκηση είναι να υλοποιήσουμε ένα 8-tap fir φίλτρο, δηλαδή σύμφωνα με τον παραπάνω τύπο ένα φίλτρο M=7 τάξης (8 όροι). Το τελικό μας κύκλωμα θα έχει την παρακάτω μορφή:



MAC: Μονάδα Πολλαπλασιασμού με Συσσωρευτή (Multiplier Accumulator Unit)

Το στοιχείο αυτό του κυκλώματος είναι υπεύθυνο για την παραγωγή των τιμών του y, εκτελώντας την πράξη:

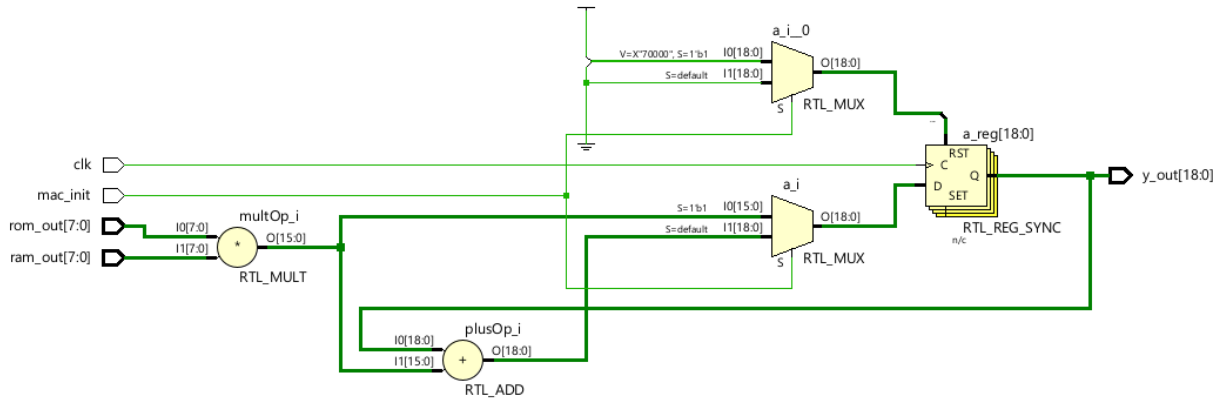
$$a \leftarrow a + b \cdot x$$

Το MAC δέχεται στην είσοδο του τους σταθερούς συντελεστές του φίλτρου και τις τιμές του σήματος εισόδου και στην έξοδό του δίνει κάθε φορά την τιμή του σήματος εξόδου y. Επιπλέον, δέχεται και ένα σήμα, το οποίο καθορίζει την αρχή της συσσώρευσης.
Σημειώνεται ότι πρέπει να προσέξουμε για την έξοδο y του φίλτρου, κάθε τιμή εξόδου να αποτελείται από 2N + logN bits, αφού η έξοδος εξαρτάται από πολλαπλασιασμούς τιμών των N bits και N προσθέσεις αριθμών των 2N bits.
Παρακάτω φαίνεται ο κώδικας σε vhdl με την υλοποίηση της μονάδας σε behavioral αρχιτεκτονική:

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

entity mac is
```

```
    Port (
           clk   : 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;
         --in order to avoid overflow we need 2N + log(N) bits
           y_out      : out STD_LOGIC_VECTOR (15 + 3  downto 0)
         );
end entity;

architecture behavioral of mac is
    signal a: std_logic_vector(15+3 downto 0);
    begin
        process (clk)
        begin
        if (clk'event and clk = '1') then
            if(mac_init = '1') then
                a <=  EXT(rom_out * ram_out, a'LENGTH);
            else
                a <= a + EXT(rom_out * ram_out, a'LENGTH);
            end if;
        end if;
        end process;
y_out <= a;
end behavioral;
```

Με χρήση του elaborated design έχουμε το παρακάτω αποτέλεσμα:



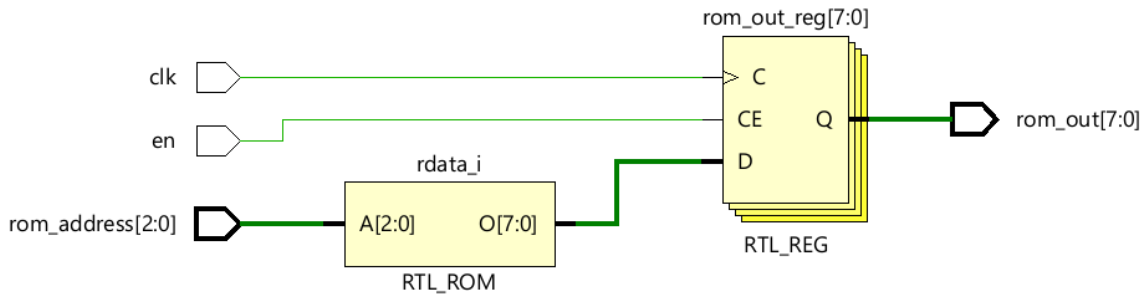
ROM

Η ROM έχει αποθηκευμένους τους σταθερούς συντελεστές του φίλτρου h. Δέχεται σαν είσοδο μια διεύθυνση και δίνει στον έξοδο την αντίστοιχη τιμή που έχει αποθηκευμένη, όταν το σήμα ενεργοποίησης της (enable) είναι 1. Οι συντελεστές του φίλτρου h είναι οι παρακάτω:
h = {1, 2, 3, 4, 5, 6, 7, 8}
Ο κώδικας σε vhdl για την περιγραφή της rom φαίνεται παρακάτω:

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

```
use ieee.NUMERIC_STD.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
           rom_address : 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"); -- initialization of rom with user data

    signal rdata : std_logic_vector(coeff_width-1 downto 0) := (others => '0');
begin
    rdata <= rom(to_integer(unsigned(rom_address)));
    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 χρήση του elaborated design εχουμε:



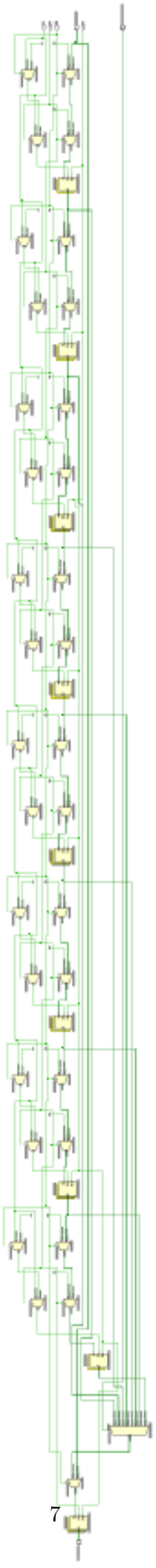
RAM

Η RAM αποθηκεύει την τελευταία τιμή του σήματος του σήματος εισόδου και τις 7 προηγούμενες τιμές που είναι απαραίτητες για τον υπολογισμό της επόμενης εξόδου. Κατά την λειτουργία ανάγνωσης δέχεται μια διεύθυνση και δίνει στην έξοδο την αντίστοιχη τιμή. Κατά την λειτουργία εγγραφής εγγράφεται η πιο πρόσφατη τιμή στην 1η θέση της ram και οι υπόλοιπες 8 ολισθαίνουν μία θέση πίσω στην ram με την τελευταία φυσικά να διαγράφεται.
Σημειώνεται ότι επειδή στην λειτουργία εγγραφής η διεύθυνση της τιμής που διαβάζεται είναι η διεύθυνση της 1ης θέσης μνήμης της RAM δίνεται κατευθείαν στην έξοδο η είσοδος και ταυτόχρονα αποθηκεύεται, καθώς αν αποθηκευόταν και μετά γινόταν προόσβαση με την διεύθυνση θα είχαμε καθυστέρηση ενός κύκλου.
Ο κώδικας σε vhdl για την περιγραφή της ram φαίνεται 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 (clk  : in std_logic;
          we   : in std_logic;                                                  --- memory write enable
          en   : in std_logic;                                        --- operation enable
          rst : in std_logic;
          ram_address : in std_logic_vector(2 downto 0);                          -- memory address
          ram_in   : in std_logic_vector(data_width-1 downto 0);                  -- input data
          ram_out   : 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'));
    signal ram_out_reg : std_logic_vector(data_width-1 downto 0);

begin


    process (clk)
    begin

        if (rst = '1') then
            for i in 0 to 7 loop
                RAM(i) <= "00000000";
            end loop;
        end if;
        if clk'event and clk = '1' then
            if en = '1' then
                if we = '1' then                                    -- write operation
                    RAM(0) <= ram_in;
                    for i in 1 to 7 loop
                        RAM(i) <= RAM(i-1);
                    end loop;
                    ram_out <= ram_in;
                else                                                    -- read operation
```

```
                    ram_out <= RAM(conv_integer(ram_address));
                end if;
            end if;
        end if;
    end process;


end Behavioral;
```

Με χρήση του elaborated design εχουμε:

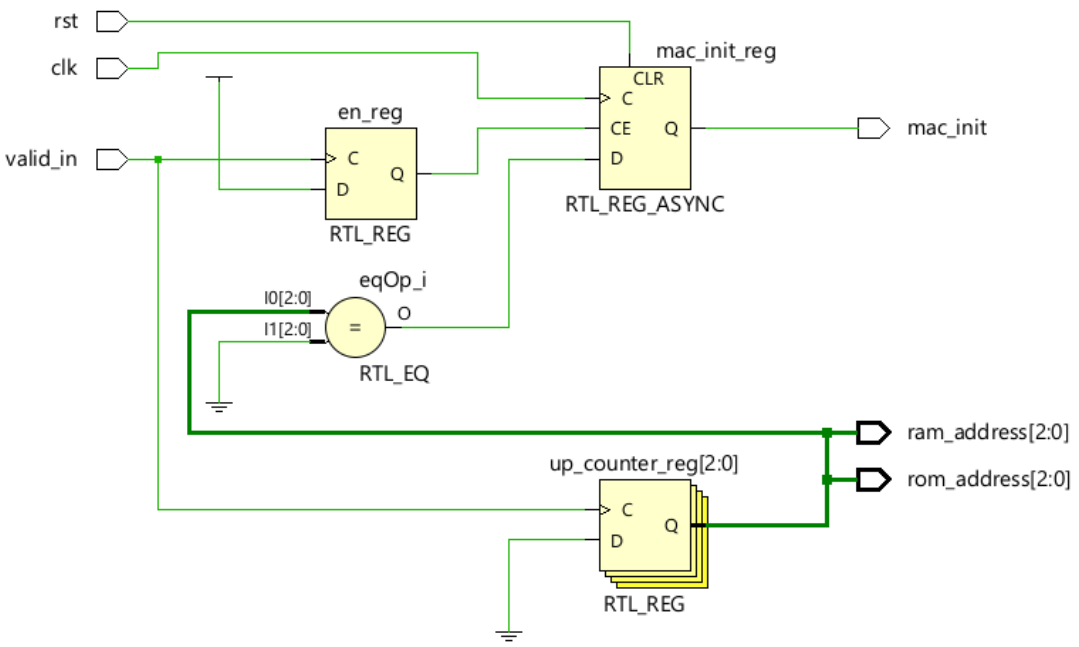


Control Unit - Μονάδα Ελέγχου

Η Μονάδα Ελέγχου είναι υπεύθυνη να δίνει τις κατάλληλες διευθύνσεις στις ram και rom σε κάθε κύκλο ρολογιού και το σήμα για αρχικοποίηση της συσσώρευσης στο MAC.
Ο κώδικας σε vhdl για την περιγραφή του control unit φαίνεται παρακάτω:

```
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 (
           clk            : in STD_LOGIC;
           rst            : in STD_LOGIC;
           valid_in       : in STD_LOGIC;
           rom_address    : out STD_LOGIC_VECTOR (2 downto 0);
           ram_address    : out STD_LOGIC_VECTOR (2 downto 0);
           mac_init       : out STD_LOGIC
        );
end entity;

architecture behavioral of control_unit is
    signal up_counter: std_logic_vector(2 downto 0);
    signal en : STD_LOGIC;
    begin
        process (clk, rst, valid_in)
        begin
        if (rst = '1') then
            up_counter <= "000";
            mac_init <= '0';
            en <= '0';
        elsif (clk'event and clk = '1') then
            if(en = '1') then
                up_counter <= up_counter +1;
                if(up_counter = "000") then
                    mac_init <= '1';
                else
                    mac_init <= '0';
                end if;
                if up_counter = "111" then
                    en <= '0';  -- <-- this line stops the MAC after 8 cycles
                end if;
            end if;
        end if;
        if(valid_in'event and valid_in = '1') then
            en <= '1';
            up_counter <= "000";
        end if;
        ram_address <= up_counter;
        rom_address <= up_counter;
        end process;

end behavioral;
```

Με χρήση του elaborated design εχουμε:



Συνολικό Κύκλωμα - FIR φίλτρο

Το συνολικό κύκλωμα συνδυάζει και συγχρονίζει τα παραπάνω επιμέρους στοιχεία. Δέχεται κάθε 8 κύκλους ρολογιού μια είσοδο, την επόμενη τιμή του σήματος εισόδου συνοδευόμενη από σήμα που επιβεβαιώνει την εγκυρότητα του και και δίνει στην έξοδο με διαφορά 8 κύκλων μεταξύ τους τις τιμές του σήματος εξόδου, οι οποίες συνοδεύνται από σήμα που δείχνει την εγκυρότητα τους, για να ξεχωρίζουν από ενδιάμεσες τιμές ή λανθασμένους υπολογισμούς.
Ο κώδικας σε vhdl για την περιγραφή του fir φίλτρου σε structural αρχιτεκτονική φαίνεται παρακάτω:

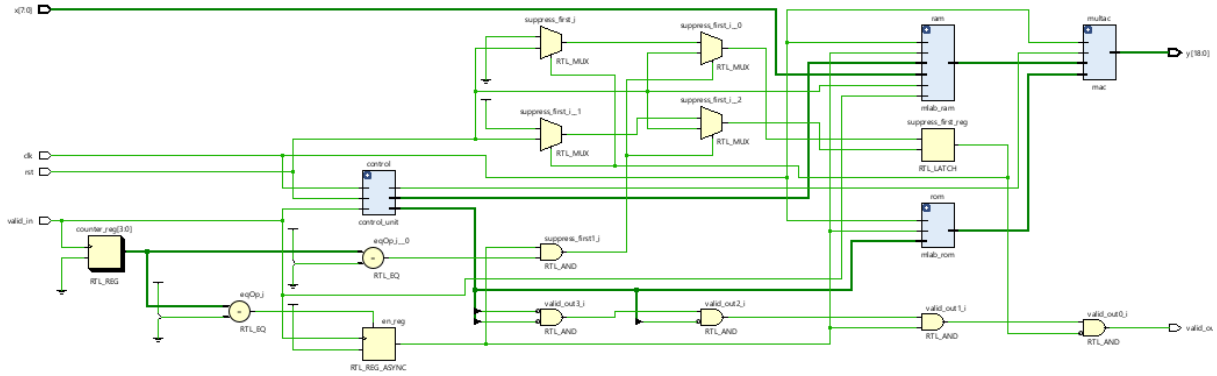
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fir_filter is
    port (
        clk : in std_logic;
        rst : in std_logic;
        valid_in : in std_logic;
        x : in std_logic_vector(8-1 downto 0);
        valid_out : out std_logic;
        y : out std_logic_vector(15 + 3  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_address    : out STD_LOGIC_VECTOR (2 downto 0);
           ram_address    : out STD_LOGIC_VECTOR (2 downto 0);
           mac_init       : out STD_LOGIC
        );
end component;

component mlab_rom is
         generic (
                coeff_width : integer :=8                          --- width of coefficients (bits)
         );
    Port ( clk : in  STD_LOGIC;
           en : in  STD_LOGIC;                                  --- operation enable
           rom_address : in  STD_LOGIC_VECTOR (2 downto 0);                        -- memory address
           rom_out : out  STD_LOGIC_VECTOR (coeff_width-1 downto 0));        -- output data
end component;

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
          ram_address : in std_logic_vector(2 downto 0);                          -- memory address
          ram_in   : in std_logic_vector(data_width-1 downto 0);                  -- input data
          ram_out   : out std_logic_vector(data_width-1 downto 0));                  -- output data
end component;

component mac is
    Port (
           clk  : 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;
        --in order to avoid overflow we need 2N + log(N) bits
           y_out      : out STD_LOGIC_VECTOR (15 + 3  downto 0)
        );
end component;

signal rom_addr, ram_addr : std_logic_vector(3-1 downto 0);
signal en : std_logic := '0';
signal rom_out, ram_out : std_logic_vector(8-1 downto 0);
signal mac_init, out_check : std_logic;
signal counter : std_logic_vector(4-1 downto 0) := "0000";
signal suppress_first : std_logic := '1';

begin

control : control_unit port map(
    clk => clk,
    rst => rst,
    valid_in => valid_in,
    rom_address => rom_addr,
```

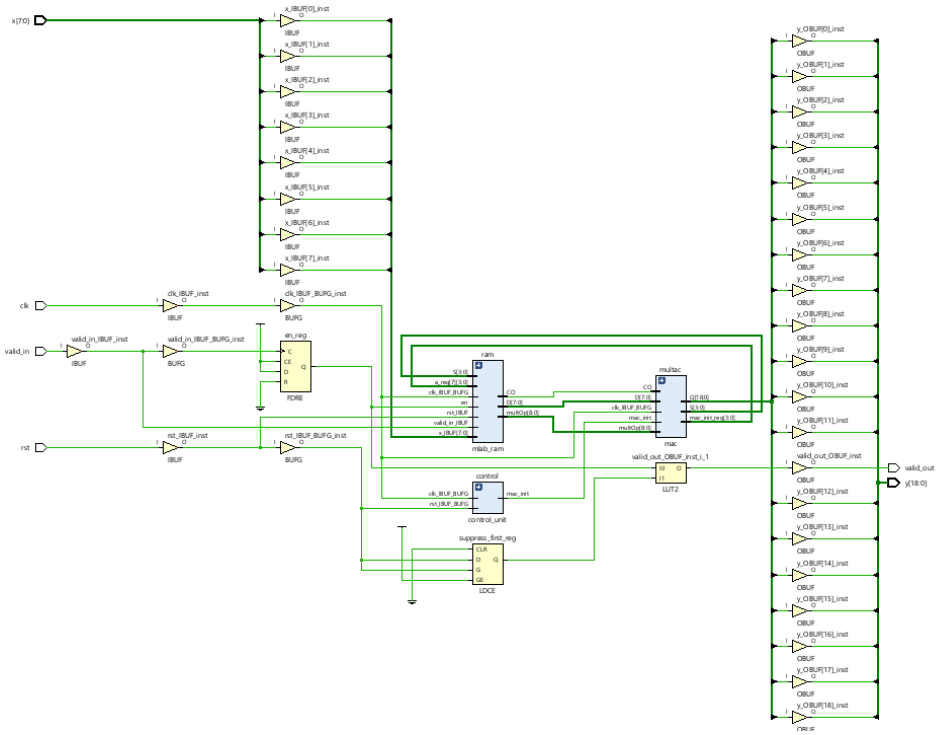
```
    ram_address => ram_addr,
    mac_init => mac_init
);

rom : mlab_rom port map(
    clk => clk,
    en => en,
    rom_address => rom_addr,
    rom_out => rom_out
);

ram : mlab_ram port map(
    clk => clk,
    rst => rst,
    we => valid_in,
    en => en,
    ram_address => ram_addr,
    ram_in => x,
    ram_out => ram_out
);

multac : mac port map(
    clk => clk,
    rom_out => rom_out,
    ram_out => ram_out,
    mac_init => mac_init,
    y_out => y
);

process(rst, clk, valid_in) begin

if (rst = '1') then
    counter <= "0000";
    en <= '0';
    valid_out <= '0';
     suppress_first <= '1';

elsif (clk'event and clk = '1') then
    counter <= counter + 1;
end if;

if (valid_in'event and valid_in = '1') then
    counter <= "0000";
    en <= '1';
end if;

 if (en = '1' and counter = "0001") then
     if (suppress_first = '1') then
         valid_out <= '0';
         suppress_first <= '0';
     end if;
 end if;

if (counter = "1001") then
```

```
    en <= '0';
    valid_out <= '0';
end if;


valid_out <= (not rom_addr(0)) and (not rom_addr(1))

and (not rom_addr(2)) and en and (not suppress_first);

end process;

end structural;
```

Με χρήση του elaborated design εχουμε:

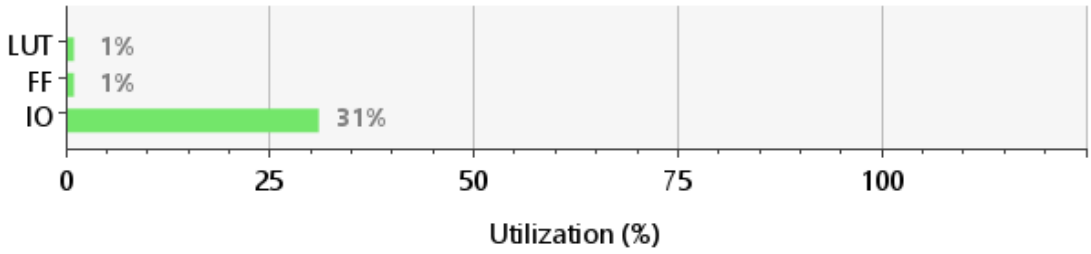


Με χρήση του synthesis εχουμε:

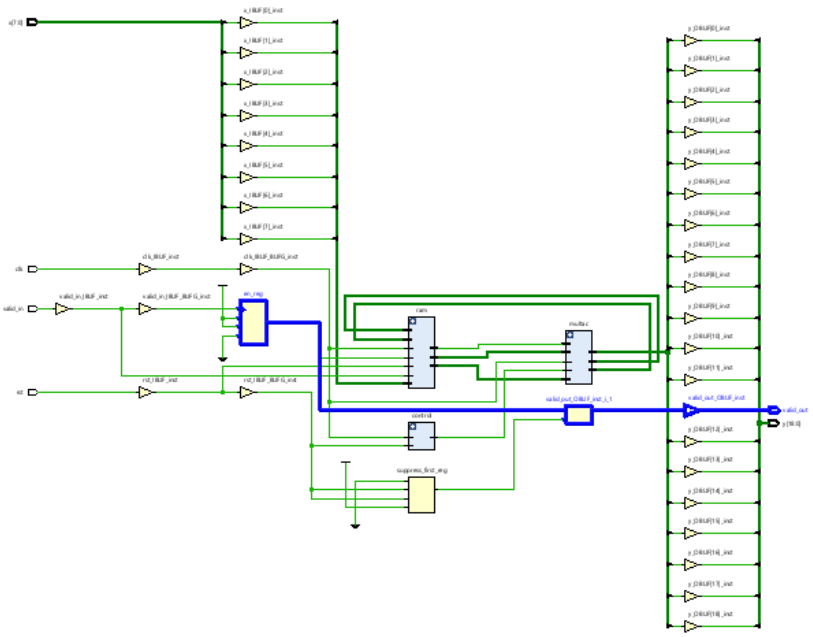


Παρακάτω φαίνοναι οι πόροι που χρησιμοποιήθηκαν, οι οποίοι παρουσιάζονται με την βοήθεια της επιλογής Utilization του synthesized design.

Resource	Utilization	Available	Utilization %
LUT	30	17600	0.17
FF	38	35200	0.11
IO	31	100	31.00



Εντοπίζουμε το παρακάτω κρίσιμο μονοπάτι με την βοήθεια του timing summary:



Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	3	3	11	en_reg/C	valid_out	4.712	3.396	1.316	∞	
Path 2	∞	8	2	2	ram/ram_out_reg[3]/C	multac/a_reg[17]/D	4.378	2.766	1.612	∞	
Path 3	∞	8	2	2	ram/ram_out_reg[3]/C	multac/a_reg[18]/D	4.286	2.674	1.612	∞	
Path 4	∞	8	2	2	ram/ram_out_reg[3]/C	multac/a_reg[16]/D	4.265	2.653	1.612	∞	
Path 5	∞	7	2	2	ram/ram_out_reg[3]/C	multac/a_reg[13]/D	4.264	2.652	1.612	∞	
Path 6	∞	7	2	2	ram/ram_out_reg[3]/C	multac/a_reg[15]/D	4.245	2.633	1.612	∞	
Path 7	∞	7	2	2	ram/ram_out_reg[3]/C	multac/a_reg[14]/D	4.172	2.560	1.612	∞	
Path 8	∞	7	2	2	ram/ram_out_reg[3]/C	multac/a_reg[12]/D	4.151	2.539	1.612	∞	
Path 9	∞	6	2	2	ram/ram_out_reg[3]/C	multac/a_reg[9]/D	4.150	2.538	1.612	∞	
Path 10	∞	6	2	2	ram/ram_out_reg[3]/C	multac/a_reg[11]/D	4.131	2.519	1.612	∞	

Παρατηρούμε ότι το κρίσιμο μονοπάτι έχει καθυστέρηση 4.712 ns. Αυτό σημαίνει ότι η μέγιστη συχνότητα λειτουργίας του fpga θα είναι ίση με τον αντίστροφο της καθυστέρησης, δηλαδή 212 MHz.

Παρακάτω φαίνεται το testbench που έχουμε κατασκευάσει για να ελέγξουμε την ορθή λειτουργία του κυκλώματός μας και στην συνέχεια παρατηρούμε τα αποτελέσματα της προσομοίωσης και σχολιάζονται τα αντίστοιχα σημεία που ελέγχουμε στο κύκλωμά μας:

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

entity fir_filter_tb is
end fir_filter_tb;

architecture behavior of fir_filter_tb is

    component fir_filter
        port (
            clk : in std_logic;
            rst : in std_logic;
            valid_in : in std_logic;
```

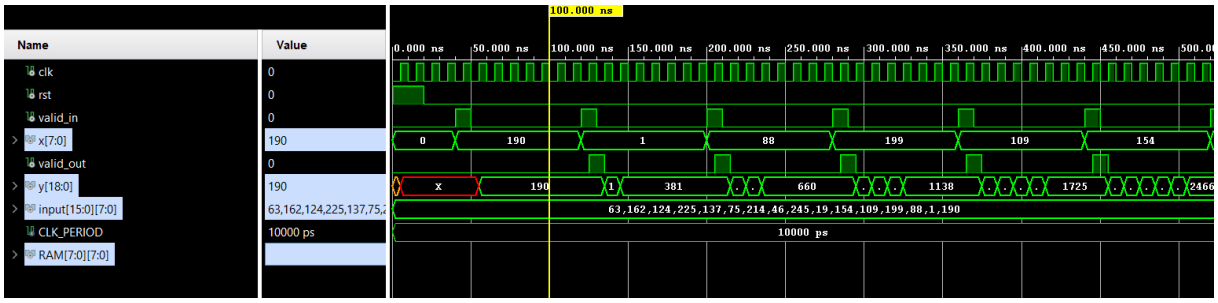
```
            x : in std_logic_vector(7 downto 0);
            valid_out : out std_logic;
            y : out std_logic_vector(18 downto 0)
        );
    end component;

    signal clk       : std_logic := '0';
    signal rst       : std_logic := '1';
    signal valid_in  : std_logic := '0';
    signal x         : std_logic_vector(8-1 downto 0) := (others => '0');
    signal valid_out : std_logic;
    signal y         : std_logic_vector(15+3 downto 0);

    type input_type is array (16-1 downto 0) of std_logic_vector (8-1 downto 0);
    signal input : input_type:= (    "00111111", "10100010", "01111100", "11100001",
    "10001001", "01001011", "11010110", "00101110",
    "11110101", "00010011", "10011010", "01101101",
    "11000111", "01011000", "00000001", "10111110");

    constant CLK_PERIOD : time := 10 ns;

begin

    uut: fir_filter port map (
        clk => clk,
        rst => rst,
        valid_in => valid_in,
        x => x,
        valid_out => valid_out,
        y => y
    );

    clk_process : process
    begin
        while now < 10 ms loop  -- simulation for 5 ms
            clk <= '0';
            wait for CLK_PERIOD / 2;
            clk <= '1';
            wait for CLK_PERIOD / 2;
        end loop;
        wait;
    end process;

stim_process : process
begin
    -- Reset sequence
    rst <= '1';
    valid_in <= '0';
    wait for 20 ns;
    rst <= '0';
    wait for 20 ns;

    -- input every 8 cycles
    for i in 0 to 15 loop
```

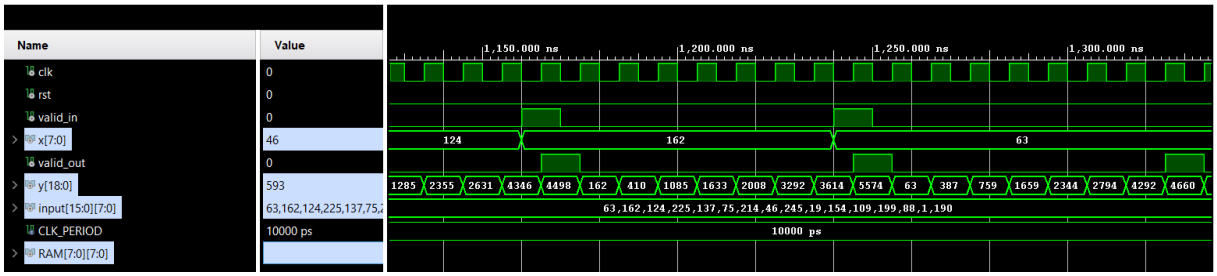
```
        valid_in <= '1';
        x <= input(i);
        wait for 10 ns;

        valid_in <= '0';
        wait for 70 ns;
    end loop;

    wait for 20 ns;
    -- reset again
    rst <= '1';
    valid_in <= '0';
    wait for 20 ns;
    rst <= '0';
    wait for 20 ns;

    -- try to input without valid in

    for i in 0 to 3 loop
        x <= input(i);
        wait for 10 ns;

        valid_in <= '0';
        wait for 70 ns;
    end loop;

    -- calculate again

    for i in 0 to 15 loop
        valid_in <= '1';
        x <= input(i);
        wait for 10 ns;

        valid_in <= '0';
        wait for 70 ns;
    end loop;


    wait;
end process;


end behavior;
```

Αρχικά, εφαρμόζεται reset στο κύκλωμα και αρχίζουν να δίνονται οι παρακάτω τιμές σήματος εισόδου:
x = {190, 1, 88, 199, 109, 154, 19, 245, 46, 214, 75, 137, 225, 124, 162, 63}

Υπολογίζουμε ότι η σωστή έξοδος είναι:
y = {190,381,660,1138,1725,2466,3226,4231,3572,4638,4995,4402,4645,4498,5574,4660}

Στα παρακάτω στιγμιότυπα από την προσομοίωση μπορούμε να παρατηρήσουμε ότι το φίλτρο δίνει στην έξοδο σωστές τιμές, με το σήμα valid_out που υποδηλώνει την σωστή έξοδο να είναι σωστά συγχρονισμένο με την σωστή τιμή εξόδου:





Στην συνέχεια εφαρμόζεται πάλι reset και αρχικά παρατηρούμε σωστό μηδενισμό όλων των τιμών της ram. Για να δοκιμάσουμε την ορθή λειτουργία του σήματος valid_in δίνουμε εισόδους χωρίς αυτό και παρατηρούμε ότι σωστά το κύκλωμα δεν ανταποκρίνεται (δεν δίνεται στην έξοδο σήμα valid_out). Τέλος, δίνονται τιμές με σωστό τρόπο και γίνονται ξανά οι υπολογισμοί που είχαμε δοκιμάσει πριν το reset και παρατηρούμε ότι το σήμα valid out δίνεται στην σωστή 1η έγκυρη τιμή.