4η Εργαστηριακή Άσκηση στο μάθημα

Ψηφιακά Συστήματα VLSI

Ομάδα 13 :

Γρίβας Αριστοτέλης – el19889

Αυγουστής Μολτσάνοβ Εμίλ – el17064

Για τη συγκεκριμένη άσκηση παραθέτουμε 2 προσεγγίσεις ως προς την υλοποίηση του κυκλώματος :

1η Προσέγγιση :

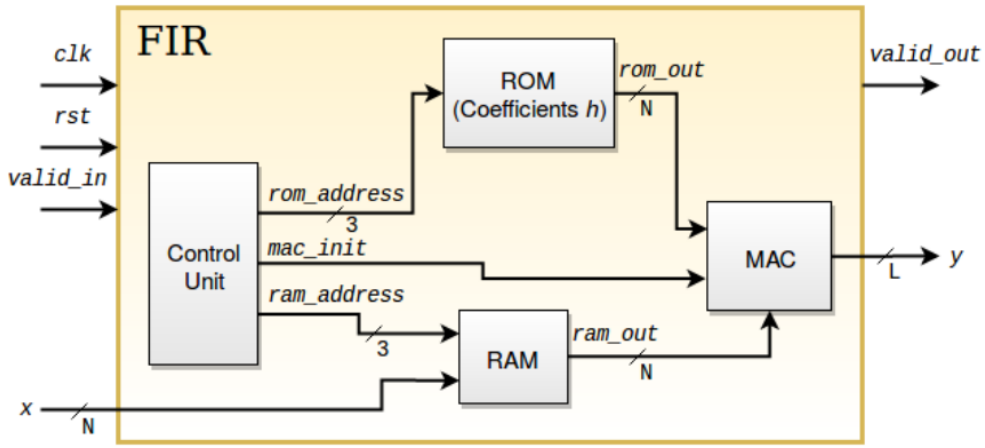
Ζητούμενο της άσκησης είναι η υλοποίηση ενός 8-tap FIR φίλτρου,του οποίου η έξοδος περιγράφεται από την παρακάτω εξίσωση :

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

όπου

- *M* είναι η **τάξη του φίλτρου**
- *y*[*n*] είναι η **έξοδος του φίλτρου** τη 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*

Η σχεδίαση του φίλτρου βασίζεται στην αρχιτεκτονική που παρουσιάζεται στην ακόλουθη εικόνα :



Τα βασικά modules της αρχιτεκτονικής του φίλτρου είναι τα ακόλουθα:

1. MAC - Μονάδα Πολλαπλασιασμού με Συσσώρευση (Multiplier Accumulator Unit): υπολογίζει την έξοδο του φίλτρου y, εκτελώντας την ακόλουθη πράξη:

a <- a + b x c

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.

2.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).

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.

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.

Ακολουθεί ο κώδικας που χρησιμοποιήθηκε για τη σχεδίαση του φίλτρου :

Για το MAC :

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

```
use IEEE.std_logic_unsigned.all;

entity mlab_mac is
    Port(CLK :IN std_logic;
         rom_in : IN std_logic_vector(7 downto 0);
         ram_in: IN std_logic_vector(7 downto 0);
         mac_init: IN std_logic;
         mac_out: OUT std_logic_vector(18 downto 0);
         valid_out : out std_logic
         );
end mlab_mac;

architecture Behavioral of mlab_mac is

signal sig1: std_logic_vector(18 downto 0) := (others=>'0');
signal sig2: std_logic_vector(18 downto 0) := (others=>'0');
signal counter: std_logic_vector(3 downto 0) := "0000";



begin




process(clk)
begin

  if mac_init = '0' then
       sig2 <= (others=>'0');
       sig1 <= (others=>'0');
```

```
        counter <= "0000";
        valid_out <= '0';
        mac_out <= sig2;

    elsif mac_init = '1'  then
      if  clk'event and clk = '1' then
        mac_out <= sig2;
        counter <= counter +1;
        sig1 <= sig1 + (rom_in * ram_in);
        valid_out <= '0';


         if counter = "1000" then
               sig2 <= sig1;
               counter <= "0000";
               sig1 <= (others=>'0');
         end if;

         if counter = "0000" then
            valid_out <= '1';
         end if;
      end if;
    end if;


end process;
end Behavioral;
```

Σχόλια : Το MAC βασίζεται στη δημιουργία μιας loop,η οποία αρχίζει να τρέχει όταν το mac_init γίνει 1.Αφού αρχίσει η loop αυτή(αρχίζει κάθε φορά όταν το Control_Unit δώσει τις διευθύνσεις “000” στις ram/rom και τελειώνει όταν δώσει τις “111”),το mac πολλαπλασιάζει τις εξόδους των ram/rom, και τις κρατά στο σήμα **sig1.**

Στους επόμενους κύκλους,παίρνει και πάλι τον πολλαπλασιασμό των εξόδων ram/rom και τους προσθέτει και πάλι στο sig1.Όταν ολοκληρωθεί η loop,και έχουν προστεθεί όλοι οι αριθμοί των ram/rom και έχουμε το επιθυμητό αποτέλεσμα στο sig1,αποθηκεύουμε την τιμή του sig1 στο σήμα sig2,δίνουμε το sig2 στην έξοδο και μηδενίζουμε το sig1 ώστε να αρχίσει στον επόμενο κύκλο να μαζεύει τους υπολογισμούς για το νέο αποτέλεσμα που θα δώσει μετά απο 9 κύκλους.

Τέλος, για reset = 0(ενεργό χαμηλά), μηδενίζονται τα sig1,sig2 καθώς και το σήμα counter που πραγματοποιεί τη loop.

<u>Για τη RAM :</u>

```
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 (reset : in std_logic;
          valid_in : 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'));

    signal dff : ram_type := (others => (others => '0'));


begin

    process (clk)

    begin

        if reset = '0' then

            RAM <= (others => (others => '0'));

        end if;


        if clk'event and clk = '1' then

            if en = '1' then

                if we = '1' then                  -- write
operation

                    if valid_in = '1' then
```

```
                        do <= di;
                        RAM  <= RAM(6 downto 0) & di;
                    else
                        do <= RAM(0);
                    end if;
                else                                -- read
operation
                    do <= RAM( conv_integer(addr));
                end if;
            end if;
        end if;
    end process;



end Behavioral;
```

Σχόλια : Η ram δίνεται υλοποιημένη μέχρι ένα βαθμό και προσθέτουμε τα εξής :

-Όταν γράφουμε νέα τιμή (valid_in = 1) ,δίνουμε στην έξοδο την τιμή της εισόδου,βάζουμε την είσοδο στην πρώτη θέση της ram και διαγράφουμε την τελευταία θέση της ram,ολισθαίνοντας τις υπόλοιπες θέσεις κατά μία.

-Όταν valid_in = 0,δίνουμε στην έξοδο την τιμή που έχει η ram στην πρώτη θέση της, ram[0], και δεν γίνεται ολίσθηση,με αποτέλεσμα να αγνοούμε εντελώς την είσοδο.

-Όταν reset = 0 (ενεργό χαμηλά), μηδενίζουμε όλες τις τιμές της ram.

-Σε οποιαδήποτε άλλη περίπτωση δίνουμε στην έξοδο την τιμή της ram στη διεύθυνση που μας υπαγορεύει το control_unit.

Για τη ROM :

```
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);

    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(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;
```

Σχόλια : Η υλοποίηση της ROM δίνεται έτοιμη .Έχει αποθηκευμένους τους συντελεστές στις σωστές θέσεις και δίνει τον κάθε έναν στην έξοδο ανάλογα με την τιμή της διεύθυνσης που δίνεται από το ControlUnit.

Για το Control Unit :

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


entity mlab_ControlUnit is
    Port(reset : IN std_logic;
         CLK: IN std_logic;
```

```
          rom_addr: OUT std_logic_vector(2 downto 0);
          ram_addr: OUT std_logic_vector(2 downto 0);
          mac_addr: OUT std_logic;
          ram_we : OUT std_logic
          );
end mlab_ControlUnit;

architecture Behavioral of mlab_ControlUnit is

signal addr_sig: std_logic_vector(2 downto 0) := "000";
signal count: std_logic_vector(3 downto 0) := "0000";

begin

process(clk)
begin

   if reset = '0' then
        count <= "0000";
        addr_sig <= "000";
        rom_addr <= "000";
        ram_addr <= "000";
        mac_addr <= '0';


   elsif clk'event and clk = '1' then

        if count = "0000" then
```

```
                ram_we <= '1';

        else
                ram_we <= '0';

        end if;

        mac_addr <= '1';
        rom_addr <= addr_sig;
        ram_addr <= addr_sig;

        addr_sig <= addr_sig + 1;
        count <= count + 1;

        if count = "1000" then
            count <= "0000";
            addr_sig <="000";
        end if;
   end if;

end process;

end Behavioral;
```

Σχόλια : Το control_unit βασίζεται σε μία loop, η οποία πραγματοποιείται μέσω του σήματος counter και σταματάει όταν φτάσει την τιμή 8,αρχίζοντας πάλι από την αρχή. Σε κάθε μία από τις καταστάσεις της loop, δίνεται στις ram/ rom η αντίστοιχη διεύθυνση,η

οποία αυξάνεται σε κάθε επανάληψη(πρώτη επανάληψη → addr = 0, δεύτερη → addr = 1 ...κ.ο.κ).

Όταν δίνεται η διεύθυνση addr=0 και αρχίζουν οι μετρήσεις,δίνονται στις εξόδους τα σήματα ram_we για εγγραφή στη ram και mac_init για αρχικοποίηση του mac για νέα μέτρηση. Τέλος για reset = 0, στέλνουμε στις ram/rom συνεχώς τις διευθύνσεις “000”.

Για το FIR :

Επίσης, έχει γίνει χρήση ενός D flip-flop για το συγχρονισμό του κυκλώματος και την αρχικοποίηση της συσσώρευσης πριν τον υπολογισμό του πρώτου y (2 κύκλοι ρολογιού για αρχικοποίηση του κυκλώματος όπως θα δούμε στο test-bench).

Τέλος ο κώδικας σε structural αρχιτεκτονική για το FIR φίλτρο,το οποίο κάνει χρήση όσον αναφέραμε :

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

entity fir is
    Port(valid_in : in std_logic;
         reset: IN std_logic;
         CLK : 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 Structural of fir is

component reg
Port ( D,CLK : IN std_logic;
       Q : OUT std_logic
     );
end component;

component mlab_ControlUnit
    Port(reset: IN std_logic;
         CLK: IN std_logic;
         rom_addr: OUT std_logic_vector(2 downto 0);
         ram_addr: OUT std_logic_vector(2 downto 0);
         mac_addr: OUT std_logic;
         ram_we : OUT std_logic
         );
end component;

component mlab_rom
      generic (
           coeff_width : integer :=8                    ---
width of coefficients (bits)
      );
    Port ( clk : in  STD_LOGIC;
                  en : in  STD_LOGIC;                   ---
operation enable
```

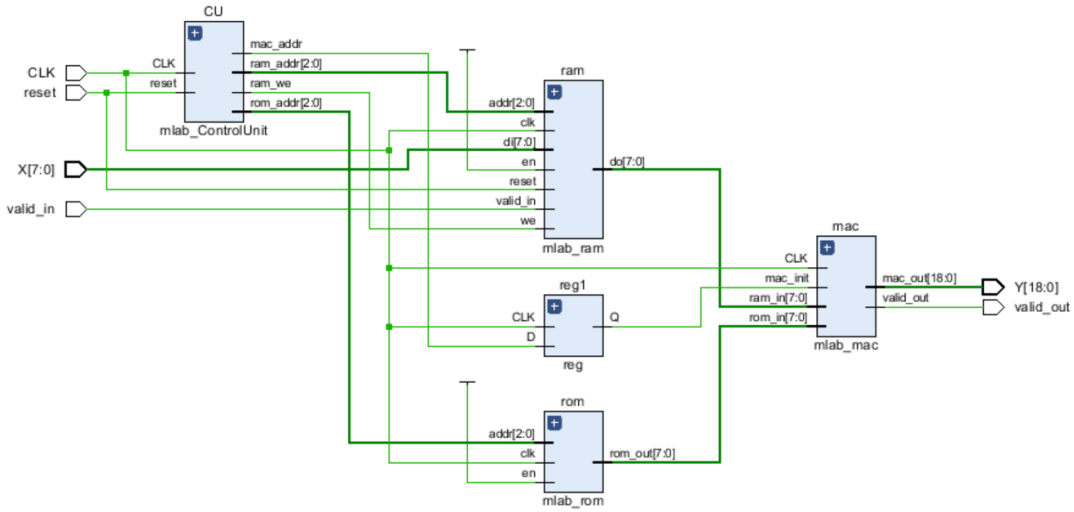
```
           addr : in  STD_LOGIC_VECTOR (2 downto 0);
      -- memory address

           rom_out : out  STD_LOGIC_VECTOR (coeff_width-1
downto 0));      -- output data

end component;


component mlab_ram

      generic (

           data_width : integer :=8                    ---
width of data (bits)

      );

    port (reset : in std_logic;

          valid_in : 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;


component mlab_mac

    Port(CLK :IN std_logic;

         rom_in : IN std_logic_vector(7 downto 0);

         ram_in: IN std_logic_vector(7 downto 0);

         mac_init: IN std_logic;

         mac_out: OUT std_logic_vector(18 downto 0);
```

```
        valid_out : out std_logic
        );
end component;

SIGNAL romin,ramin : std_logic_vector(2 downto 0);
SIGNAL romin1,ramin1 : std_logic_vector(2 downto 0);
SIGNAL macin,ramwe,ramwe1,macout,macout1,valid_out_sig:
std_logic;
SIGNAL romout,ramout : std_logic_vector(7 downto 0);

begin

CU : mlab_ControlUnit port
map(reset,CLK,romin,ramin,macin,ramwe);

rom : mlab_rom port map(CLK,'1',romin,romout);

ram : mlab_ram port
map(reset,valid_in,CLK,ramwe,'1',ramin,X,ramout);

mac : mlab_mac port map(CLK,romout,ramout,macout,Y,valid_out);

reg1: reg port map(macin,CLK,macout);

end Structural;
```

Σχόλια : Παρατηρούμε πως το κύκλωμα αποτελείται από τη σύνδεση των κυκλωμάτων που παρουσιάστηκε παραπάνω,καθώς και του

D-flipflop ,που χρησιμοποιείται για το σήμα αρχικοποίησης του mac, mac_init, το οποίο εξασφαλίζει πως οι πράξεις στο mac θα αρχίσουν όταν θα έχουν αρχικοποιηθεί οι ram/rom και θα είναι έτοιμες να δώσουν αριθμό στην έξοδό τους.

Ακολουθεί το rtl διάγραμμα :



Παρατηρούμε πως το RTL διάγραμμα συμφωνεί με αυτό που δόθηκε στην εκφώνηση της άσκησης,με τη διαφορά του D-flipflop που προσθέσαμε.

Το testbench που χρησιμοποιήθηκε για την επαλήθευση των αποτελεσμάτων δίνεται ως εξής :

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.std_logic_unsigned.all;


entity fir_testbench is

end fir_testbench;
```

```
architecture Behavioral of fir_testbench is

SIGNAL valid_in : std_logic:= '1';
SIGNAL valid_out : std_logic;
SIGNAL reset : std_logic:= '1';
SIGNAL clk : std_logic;
SIGNAL X : std_logic_vector(7 downto 0) := "00000001";
SIGNAL Y : std_logic_vector(18 downto 0);

begin
UUT : entity work.fir port
map(valid_in,reset,clk,X,Y,valid_out);

lk_process :process
   begin

        clk <= '0';
        wait for 2ns;
        clk <= '1';
        wait for 2ns;

end process;
```

```
reset_process :process
   begin
        reset <= '1';
        wait for 800ns;
        reset <= '0';
        wait for 60ns;

end process;

valid_process :process
   begin
        valid_in <= '1';
        wait for 670ns;
        valid_in <= '0';
        wait for 130ns;

end process;


 proc2 : process is
 begin


 x <= "10100111";
 wait for 40ns;
 x <= "00001001";
 wait for 36ns;
```

```
x <= "11011001";
wait for 36ns;
x <= "11101111";
wait for 36ns;
x <= "10101101";
wait for 36ns;
x <= "11000001";
wait for 36ns;
x <= "10111110";
wait for 36ns;
x <= "01100100";
wait for 36ns;
x <= "10100111";
wait for 36ns;
x <= "00101011";
wait for 36ns;
x <= "10110100";
wait for 36ns;
x <= "00001000";
wait for 36ns;
x <= "01000110";
wait for 36ns;
x <= "00001011";
wait for 36ns;
x <= "00011000";
wait for 36ns;
x <= "11010010";
wait for 36ns;
x <= "10110001";
```

```
 wait for 36ns;
 x <= "01010001";
 wait for 36ns;
 x <= "11110011";
 wait for 36ns;
 x <= "00001000";

 wait for 36ns;
 end process proc2;

end Behavioral;
```

Και ακολουθούν οι κυματομορφές που παράγονται μέσω της προσομοίωσης :

Στις κυματομορφές αυτές έχουμε χρησιμοποιήσει τις τιμές που δόθηκαν στο μάθημα κατά τη διάρκεια της εξέτασης. Παρατηρούμε πως τα αποτελέσματα είναι αυτά που θα περιμέναμε και δίνονται κάθε 9 κύκλους ρολογιού σύμφωνα με τον τύπο του φίλτρου,ενώ για τις επιλογές valid_in = 0,reset = 0(το reset ενεργοποιείται χαμηλά),η έξοδος μας είναι η αναμενόμενη,δηλαδή σταθερή για valid_in = 0,ενώ για rst = 0 μηδενίζεται και αρχίζει η μέτρηση από την αρχή με τη ram αρχικοποιημένη με μηδενικά.

Τέλος παραθέτουμε τους πόρους που χρησιμοποιεί το κύκλωμα μας ,αφού έχουμε τρέξει τη σύνθεση :

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	Bonded IOB (100)	BUFGCTRL (32)
fir	182	189	8	31	1
CU (mlab_ControlUnit)	14	12	0	0	0
mac (mlab_mac)	73	100	0	0	0
ram (mlab_ram)	60	72	8	0	0
reg1 (reg)	11	1	0	0	0
rom (mlab_rom)	25	4	0	0	0

2η Προσέγγιση :

Control Unit

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

entity control is
port(
clk,rst,en : in std_logic;
address : out std_logic_vector(2 downto 0);
mac, to_tff, to_control : out std_logic);
end control;

architecture Behavioral of control is

signal intr : std_logic_vector(2 downto 0);
signal mac_intr : std_logic;

begin

proc: process(intr,clk,rst,en) is
begin

  if (rst='1') then intr <= "000";
```

```
  else
    if(rising_edge(clk)) then
      if(en='1') then
        if(intr="111") then intr <= "000";
        else intr <= intr+"001";
        end if;
      end if;
    end if;
  end if;

address <= intr;
mac_intr <= ((not intr(2))and(not intr(1))and(not intr(0)));
mac <= mac_intr; --1 if counter=000, 0 if else--
to_tff <= not(intr(2) and intr(1) and intr(0));
to_control <= not(mac_intr);  --0 if counter=0, 1 if else--

end process proc;
end Behavioral;
```

Σχόλια:

Το control unit είναι απλώς ένας μετρητής άνω των τριών bit, που μετράει από 0 έως 8, έπειτα επιστρέφει στο 0 και ξεκινάει να μετράει άνω από την αρχή. Η έξοδος address είναι 3bit και δείχνει την ένδειξη του μετρητή. Η έξοδος mac είναι 1bit και παίρνει την τιμή 1 μόνο όταν address=000, διαφορετικά mac=0. Η έξοδος to_control είναι 1bit και παίρνει την τιμή 0 μόνο όταν address=000, διαφορετικά to_control=1. Η

έξοδος to_tff είναι 1bit και παίρνει την τιμή 0 μόνο όταν address=111, διαφορετικά to_control=1.

Mac:

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

entity mac is
port(
init,clk,rst : in std_logic;
x,h  : in std_logic_vector(7 downto 0);
y_out    : out std_logic_vector(18 downto 0));
end mac;

architecture Behavioral of mac is

component dff is
port(
D       : in std_logic_vector(18 downto 0);
clk,rst     : in std_logic;
Q       : out std_logic_vector(18 downto 0));
end component;

signal y_inter : std_logic_vector(18 downto 0);
signal p : std_logic_vector(15 downto 0);
```

```
signal sum,sig,gate: std_logic_vector(18 downto 0);

begin

with init select

gate <= "0000000000000000000" when '1',
        "1111111111111111111" when '0',
        "0000000000000000000" when others;



p <= x * h;
sum <= p+y_inter;
sig <= sum and gate;

d1: dff port map(clk => clk, D => sig, Q => y_inter, rst=>rst);

y_out <= y_inter;

end Behavioral;
```

Σχόλια:

Με βάση τον κώδικα όταν η είσοδος init γίνει 1, η έξοδος y θα γίνει 0 με το που έρθει η θετική ακμή του ρολογιού. Επίσης η έξοδος είναι 19 bit, επειδή με βάση την εξίσωση της συνέλιξης, αν υποθέσουμε ότι όλοι οι όροι της εισόδου x και της συνάρτησης μεταφοράς h παίρνουν την

μεγαλύτερη δυνάτη τιμή, δηλαδή την 11111111, τότε y= 111 1111 0000 0000 1000, δηλαδή η έξοδος χρειάζεται 19bit για να αναπαρασταθεί.

RAM:

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

entity mlab_ram is
port (
rst  : in std_logic;
clk  : in std_logic;
we   : 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 mlab_ram;

architecture Behavioral of mlab_ram is

type ram_type is array (7 downto 0) of std_logic_vector (7 downto 0);
signal RAM : ram_type := (others => (others => '0'));
signal intr : std_logic_vector(7 downto 0);
begin

intr <= RAM(conv_integer(addr));
```

```
 process (rst,clk,we,di,intr)
 begin

 if(rst='1') then
   RAM(7) <= "00000000";
   RAM(6) <= "00000000";
   RAM(5) <= "00000000";
   RAM(4) <= "00000000";
   RAM(3) <= "00000000";
   RAM(2) <= "00000000";
   RAM(1) <= "00000000";
   RAM(0) <= "00000000";
 else
   if (rising_edge(clk)) then
     if (we = '1') then
       RAM <= (RAM(6 downto 0))&di;
       do <= di;
     else
       do <= intr;
     end if;
   end if;
 end if;

 end process;

end Behavioral;
```

ROM:

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


entity mlab_rom is
Port(
clk : in  STD_LOGIC;
addr : in  STD_LOGIC_VECTOR (2 downto 0);
rom_out : out  STD_LOGIC_VECTOR (7 downto 0));
end mlab_rom;

architecture Behavioral of mlab_rom is

type rom_type is array (7 downto 0) of std_logic_vector (7 downto 0);
signal rom : rom_type:= ("00001000", "00000111", "00000110",
"00000101", "00000100", "00000011", "00000010", "00000001");

signal rdata : std_logic_vector(7 downto 0) := (others => '0');

begin

rdata <= rom(conv_integer(addr));

  process (clk)
```

```
  begin
    if (rising_edge(clk)) then
      rom_out <= rdata;
    end if;
  end process;

end Behavioral;
```

Dff:

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

entity dff is
port(
D       : in std_logic_vector(18 downto 0);
clk,rst     : in std_logic;
Q       : out std_logic_vector(18 downto 0));
end dff;

architecture Behavioral of dff is
begin

proc : process(D,clk,rst) is
begin

  if(rst='1') then Q<="0000000000000000000";
```

```
  else
    if(rising_edge(clk)) then Q <= D;
    end if;
  end if;

end process proc;

end Behavioral;
```

Σχόλια: Είναι απλώς ένας 19bit καταχωρητής.

Dff_1bit:

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

entity dff_1bit is
port(
D,rst,clk : in std_logic;
Q : out std_logic);
end dff_1bit;

architecture Behavioral of dff_1bit is

begin

proc1 : process(D,rst,clk) is
begin
```

```
  if(rst='1') then Q <='0';
  else
    if(rising_edge(clk)) then Q <= D;
    end if;
  end if;
end process proc1;

end Behavioral;
```

Σχόλια: Είναι απλώς ένα 1bit D flip flop.

Tff:

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

entity tff is
port(
T,rst,clk : in std_logic;
Q : out std_logic);
end tff;

architecture Behavioral of tff is
signal intr : std_logic;

begin

proc1 : process(T,rst,clk) is
begin
```

```
  if(rst='1') then intr <='0';
  else
    if(rising_edge(clk)) then intr <= intr xor T;
    end if;
  end if;
end process proc1;

Q <= intr;

end Behavioral;
```

Σχόλια: Είναι απλώς ένα 1bit T flip flop.

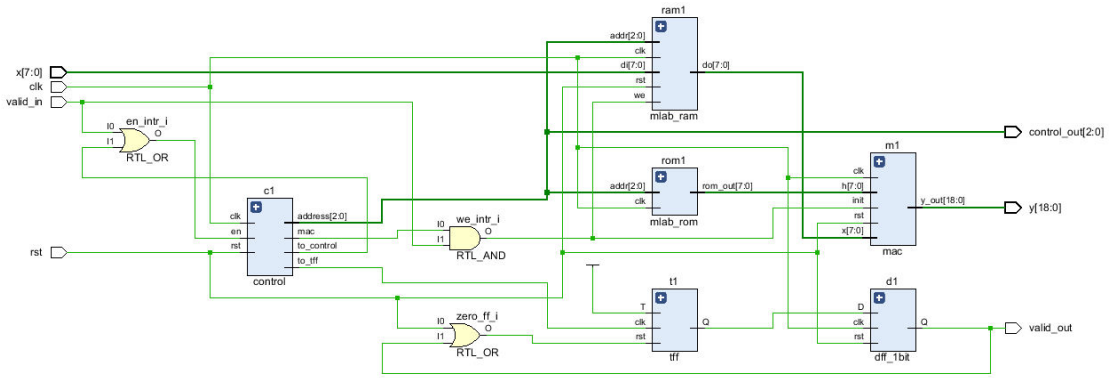
FIR φίλτρο:

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

entity FIR is
port(
clk,rst,valid_in: in std_logic;
x : in std_logic_vector(7 downto 0);
valid_out: out std_logic;
control_out : out std_logic_vector(2 downto 0);
y : out std_logic_vector(18 downto 0));
end FIR;
```

```
architecture Behavioral of FIR is

component dff_1bit is
port(
D,rst,clk : in std_logic;
Q : out std_logic);
end component;

component tff is
port(
T,rst,clk : in std_logic;
Q : out std_logic);
end component;

component mac is
port(
init,clk,rst : in std_logic;
x,h  : in std_logic_vector(7 downto 0);
y_out    : out std_logic_vector(18 downto 0));
end component;

component control is
port(
clk,rst,en : in std_logic;
address : out std_logic_vector(2 downto 0);
mac, to_tff, to_control : out std_logic);
end component;
```

```
component mlab_ram is
port (
rst  : in std_logic;
clk  : in std_logic;
we   : 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 mlab_rom is
Port(
clk : in  STD_LOGIC;
addr : in  STD_LOGIC_VECTOR (2 downto 0);
rom_out : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

signal
zero_ff,valid_and,valid_out_intr,mac_intr,clk_tff,we_intr,tff_to_dff,
en_intr, not_zero : std_logic;
signal address_intr : std_logic_vector(2 downto 0);
signal ram_out,rom_out_intr : std_logic_vector(7 downto 0);
begin

valid_and <= valid_in and valid_out_intr;
we_intr <= mac_intr and valid_in; -- read input if (counter =0 and
valid_in=1)
```

```
zero_ff <= rst or valid_out_intr;  --reset ffs if rst=1 or if valid_out=1

en_intr <= valid_in or not_zero;  --enable counter if valid_in=1 or if
counter/=0


c1: control port map(clk=>clk, rst=>rst, en => en_intr,
address=>address_intr, mac=>mac_intr,to_tff=>clk_tff, to_control =>
not_zero);

t1: tff port map(clk=> clk_tff, T=>'1',rst=>zero_ff, Q=> tff_to_dff);

d1: dff_1bit port map(clk => clk, rst=>rst, D => tff_to_dff, Q =>
valid_out_intr);


valid_out <= valid_out_intr;



ram1: mlab_ram port map(rst=>rst, clk=>clk, we=>we_intr,
addr=>address_intr,di=>x,do=>ram_out);

rom1: mlab_rom port map(clk=>clk,
addr=>address_intr,rom_out=>rom_out_intr);

m1  : mac port map(clk=>clk, rst=>rst, init=>we_intr, x=>ram_out,
h=>rom_out_intr, y_out=>y );


control_out <= address_intr;


end Behavioral;
```



Πάνω φαίνεται το rtl schematic του φίλτρου. Με βάση αυτό θα εξηγήσουμε τη λειτουργία του κυκλώματος.

Καταρχάς όσον αφορά το control unit, βλέπουμε πως για την είσοδο enable ισχύει: en=(valid_in) or (to_control) , οπότε στη θετική ακμή του ρολογιού θα έχουμε μέτρηση άνω μόνο όταν valid_in=1 ή όταν η ένδειξη του μετρητή είναι διάφορη του 0. Άρα όταν η ένδειξη του είναι 0 και ταυτόχρονα valid_in=0, τότε η ένδειξη του μετρητή δεν θα αλλάζει στις θετικές ακμές του ρολογιού αλλά θα παραμένει σταθερη, δηλαδή 0. Με αυτόν τον τρόπο εξασφαλίζεται ότι ακόμα και αν δύο διαδοχικά valid_in απέχουν απόσταση πάνω από 8 κύκλους ρολογιού, ο μετρητής δεν θα ξεκινήσει καινούρια μέτρηση από 0 έως 8 αλλά η έξοδος θα παραμένει σταθερή στο 0. Και επειδή η ένδειξη του μετρητή του control unit είναι ο δείκτης address της rom και της ram, όσο δεν έρθει το επόμενο valid_in=1, δεν θα ξεκινήσει καινούργιος υπολογισμός του y.

Τώρα για την ram, παρατηρούμε ότι we=(mac) and (valid_in), οπότε μόνο όταν valid_in=1 και ταυτόχρονα η ένδειξη του μετρητή είναι 0, τότε στη θετική ακμή του ρολογιού θα γίνει εγγραφή στην ram της καινούργιας εισόδου x.

Επίσης για την mac, παρατηρούμε ότι init=(mac) and (valid_in), οπότε μόνο όταν valid_in=1 και ταυτόχρονα η ένδειξη του μετρητή είναι 0, τότε στη θετική ακμή του ρολογιού η έξοδος y θα αρχικοποιηθεί στο 0 και θα ξεκινήσει ο υπολογισμός του y λαμβάνοντας πια υπόψη και την πιο φρέσκια τιμή του x.

Τέλος, το valid_out υπολογίζεται με τον εξής τρόπο: με βάση το κύκλωμα η έξοδος y παίρνει 8 παλμούς ρολογιού να υπολογιστεί. Σε αυτούς τους 8 παλμούς το control unit εκτελεί έναν πλήρη κύκλο μετρήσεων απο 0 έως 8. Οπότε αρκεί το valid_out να γίνει 1 όταν ολοκληρωθεί αυτός ο κύκλος. Θεωρώντας τον κύκλο αυτόν ως συμβάν, μπορούμε να μετρήσουμε αυτό το συμβάν με έναν ασύγχρονο μετρητή του 1 bit, το οποίο το υλοποιούμε με ένα T flip flop, το ρολόι του οποίου το συνδέουμε στην έξοδο to_tff του control unit. Η έξοδος του T flip flop

συνδέεται στην είσοδο ενός D flip flop ξεκάθαρα για λόγους συγχρονισμού, επειδή θέλουμε το valid_out να γίνει 1 ένα παλμό ρολογιού αφού η έξοδος του T flip flop γίνει 1. Και επειδή θέλουμε valid_out=1 μόνο για έναν παλμό ρολογιού, με το που δούμε valid_out=1, αρχικοποιούμε ασύγχρονα την έξοδο του T flip flop στο 0 μέσω του valid_out, οπότε εξαιτίας της μεσολάβησης του D flip flop, το valid_out γίνεται 0 όχι ακαριαία αλλά στον επόμενο παλμό ρολογιού, που είναι και το επιθυμητό.

Testbench:

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

entity tb_fir is
end tb_fir;

architecture Behavioral of tb_fir is

component FIR is
port(
clk,rst,valid_in: in std_logic;
x : in std_logic_vector(7 downto 0);
control_out : out std_logic_vector(2 downto 0);
valid_out: out std_logic;
y : out std_logic_vector(18 downto 0));
end component;

signal clk,valid_in,valid_out: std_logic :='0';
```

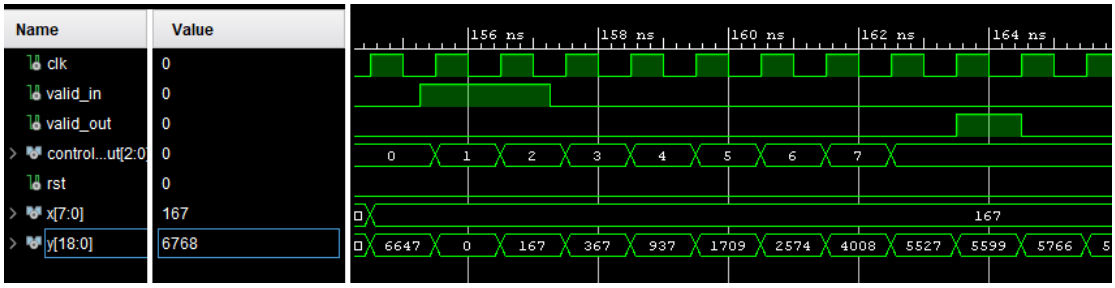
```
--
--signal clk,valid_out: std_logic :='0';
--signal valid_in: std_logic :='1';
--

signal control_out : std_logic_vector(2 downto 0) :="000";
signal rst : std_logic :='0';
signal x : std_logic_vector(7 downto 0) :="00000000";
signal y : std_logic_vector(18 downto 0) :="0000000000000000000";

begin

t1: FIR port map(clk=>clk, rst=>rst, valid_in=>valid_in, x=>x,
valid_out=>valid_out, y=>y, control_out => control_out);

proc1 : process is
begin



clk <= '0';
wait for 0.5ns;
clk <= '1';
wait for 0.5ns;

end process proc1;

proc3 : process is
```

```
begin

wait for 0.01ns;
rst <= '1';
wait for 0.5ns;
rst <= '0';
wait for 50000ns;

end process proc3;

proc4 : process is
begin

wait for 3.25ns;
valid_in <= '1';
wait for 2ns;
valid_in <= '0';
wait for 8.75ns;
wait for 5ns;

end process proc4;

proc2 : process is
begin

wait for 2.5ns;
```

```
x <= "10100111";
wait for 19ns;
x <= "00001001";
wait for 19ns;
x <= "11011001";
wait for 19ns;
x <= "11101111";
wait for 19ns;
x <= "10101101";
wait for 19ns;
x <= "11000001";
wait for 19ns;
x <= "10111110";
wait for 19ns;
x <= "01100100";
wait for 19ns;
x <= "10100111";
wait for 19ns;
x <= "00101011";
wait for 19ns;
x <= "10110100";
wait for 19ns;
x <= "00001000";
wait for 19ns;
x <= "01000110";
wait for 19ns;
x <= "00001011";
```

```
wait for 19ns;
x <= "00011000";
wait for 19ns;
x <= "11010010";
wait for 19ns;
x <= "10110001";
wait for 19ns;
x <= "01010001";
wait for 19ns;
x <= "11110011";
wait for 19ns;
x <= "00001000";

wait for 16.5ns;


end process proc2;

end Behavioral;
```

Simulation:



Στο παραπάνω κομμάτι του simulation φαίνεται πως αφού το valid_in γίνει 1 και έρθει η θετική ακμή ρολογιού, χρειάζεται να περάσουν 8

παλμοί για να εμφανιστεί ορθή τιμή της y, που την επιβεβαιώνουμε μέσω του σήματος valid_out.

Τέλος, αφού εκτελέσουμε synthesis τρέχουμε το report utilization και βρίσκουμε τους ακόλουθους πόρους του FPGA που χρησιμοποιούνται.

Name 1	Slice LUTs (17600)	Slice Registers (35200)	Bonded IOB (100)	BUFGCTRL (32)
N FIR	88	100	34	1
c1 (control)	27	3	0	0
d1 (dff_1bit)	1	1	0	0
m1 (mac)	12	19	0	0
ram1 (mlab_ram)	25	72	0	0
rom1 (mlab_rom)	23	4	0	0
t1 (tff)	1	1	0	0