2024

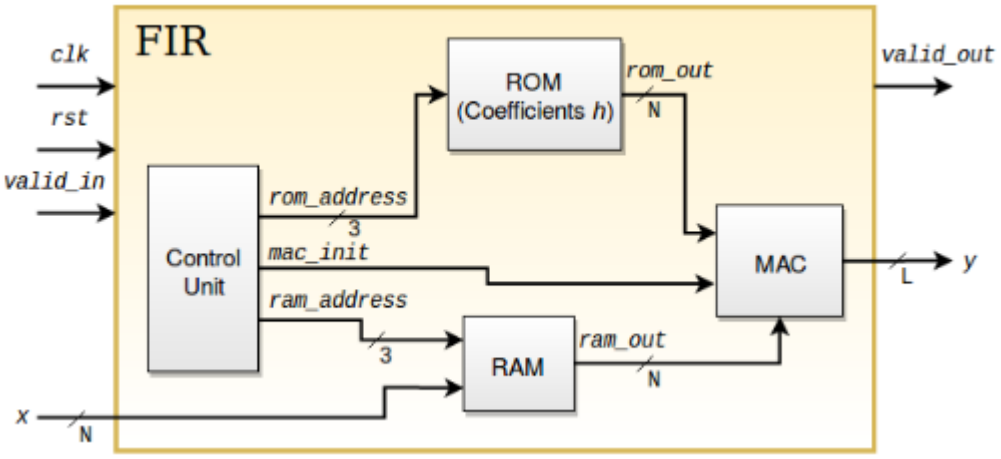
Αναφορά 4ης εργαστηριακής Άσκησης

Εργαστήριο VLSI

ΠΑΠΑΔΟΠΟΥΛΟΣ ΣΠΥΡΙΔΩΝ (ΑΜ):03120033
ΕΜΜΑΝΟΥΗΛ ΞΕΝΟΣ (ΑΜ):03120850

Ζήτημα 1ο: Υλοποίηση FIR

Για την υλοποίηση του FIR θα χρησιμοποιήσουμε την προτεινόμενη αρχιτεκτονική που δίνεται στην εκφώνηση της άσκησης. Η αρχιτεκτονική αυτή περιλαμβάνει ένα MAC unit που υλοποιεί την πράξη $y_n = y_{n-1} + x \cdot h$, μία ROM που είναι αποθηκεύμενοι οι συντελεστές του φίλτρου h, μία RAM που αποθηκεύεται η τωρινή τιμή του x και οι 7 προηγούμενες τιμές του χ και ένα control unit που συγχρονίζει τα παραπάνω components. Μία εικόνα που δείχνει την σύνδεση των παραπάνω είναι η ακόλουθη:



Αρχικά θα δείξουμε την υλοποίηση των παραπάνω components σε VHDL και στην συνέχεια θα τα συνδέσουμε σε ένα component που υλοποιεί την πράξη $y[n] = \sum_{k=0}^{M} h[k]x[n-k]$. Θα υλοποιήσουμε ένα 8-tap FIR φίλτρο, δηλαδή M=7. Εδώ αξίζει να αναφέρουμε ότι η RAM και η ROM μας είναι zero indexed, δηλαδή οι διευθύνσεις τους ξεκινάνε από το 0.

MAC Unit

Ο κώδικας για το MAC είναι ο ακόλουθος:

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

entity MAC is
    Port (
        x, h: in std_logic_vector(7 downto 0);
        mac_init: in std_logic;
        clk: in std_logic;
        y: out std_logic_vector(17 downto 0)
    );
end MAC;

architecture Behavioral of MAC is
    signal sum: unsigned(17 downto 0) := (others => '0');
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if mac_init = '1' then
                sum <= "0000000000000000"+unsigned(x) * unsigned(h);
            else
                sum <= sum + unsigned(x) * unsigned(h);
```

```
            end if;
            y <= std_logic_vector(sum);
        end if;
    end process;

end Behavioral;
```

Στον παραπάνω κώδικα βλέπουμε ότι το MAC έχει ως είσοδο τους συντελεστές x[7-i] και h[i] με τα οποία θα γίνει ο πολλαπλασιασμός, το mac_init που όταν γίνει 1 γίνεται reset η έξοδος του έξοδος y, το ρολόι για τον συγχρονισμό του κυκλώματος και ως έξοδο έχει μόνο την έξοδο y. Στην υλοποίησή του έχουμε ένα σήμα που κρατάμε το προσωρινό άθροισμα των συντελεστών και δημιουργούμε ένα process που έχει μόνο το ρολόι στο sensitivity list. Εντός του process σε περίπτωση που έχουμε rising edge του ρολογιού αρχικά ελέγχουμε αν το mac_init είναι 1. Σε περίπτωση που είναι 1 τότε θέτουμε το sum ως το γινόμενο των τωρινών x και h που έχουμε στην είσοδο, αλλιώς αν το mac_init δεν είναι 1 τότε θέτουμε το sum ως το άθροισμα του sum έως τώρα με το γινόμενο των τωρινών x και h. Τέλος θέτουμε το y ίσο με το sum. Έτσι βλέπουμε ότι όταν το mac_init δεν είναι 1 ισχύει ότι $y_\kappa = y_{\kappa-1} + x \cdot h$, ενώ όταν το mac_init είναι 1 το y είναι $y = x \cdot h$. Προκειμένου λοιπόν το MAC να υλοποιεί την πράξη $y[n] = \sum_{k=0}^{M} h[k]x[n-k]$ πρέπει να στέλνουμε τα κατάλληλα δεδομένα από την RAM και την ROM και να αρχικοποιούμε την διαδικασία με κατάλληλο χρονισμό χρησιμοποιώντας το MAC init.

ROM

Η υλοποίηση της ROM μας δόθηκε έτοιμη από την εκφώνηση της άσκησης. Ο VHDL κώδικάς της είναι:

```
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;
```

Σε αυτή βλέπουμε ότι η ROM έχει ως είσοδο το ρολόι, ένα σήμα enable, η διεύθυνση του δεδομένου που θέλουμε να εξάγουμε και η έξοδος είναι το δεδομένο που θέλουμε να εξάγουμε. Στην υλοποίηση βλέπουμε ότι έχει οριστεί ένας πίνακας με όνομα rom μεγέθους 8 με τους συντελεστές του h και ένα σήμα rdata που θα θέσουμε προσωρινά τα δεδομένα που εξάγουμε από την ROM. Στην συνέχεια θέτουμε το rdata να είναι ίσο με το στοιχείο του πίνακα rom με διεύθυνση ίση με με το σήμα addr που λαμβάνουμε ως είσοδο. Επειδή οι πίνακας γίνονται index με integers μετατρέπουμε το addr σε integer γιατί το addr είναι σε binary μορφή. Στην συνέχεια υπάρχει ένα process με sensitivity list το ρολόγοι και ελέγχουμε αν το clock είναι σε rising edge και αν το enable είναι 1. Σε περίπτωση που ισχύουν και τα δύο τότε θέτουμε την έξοδο της rom να είναι ίση με rdata.

RAM

Για την υλοποίηση της RAM μας δίνεται η υλοποίηση μίας write-first μνήμης. Σε αυτή την υλοποίηση θα πρέπει να προσθέσουμε ένα κομμάτι κώδικα που υλοποιεί την μετατόπιση των δεδομένων των γίνεται μία 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 (clk  : in std_logic;
          we   : in std_logic;                      --- memory write enable
             en   : in std_logic;                --- operation enable
             reset: in std_logic;
          addr : in std_logic_vector(2 downto 0);           -- memory address
          di   : in std_logic_vector(data_width-1 downto 0);        -- input data
          do   : out std_logic_vector(data_width-1 downto 0));      -- output data
end mlab_ram;

architecture Behavioral of mlab_ram is

    type ram_type is array (7 downto 0) of std_logic_vector (data_width-1 downto 0);
    signal RAM : ram_type := (others => (others => '0'));

begin


    process (clk)
    begin
        if clk'event and clk = '1' then
            if en = '1' then
                if reset='1' then
                    for i in 0 to 7 loop
                        RAM(i)<="00000000";
```

```
                end loop;
                do<="00000000";
            else
                if we = '1' then                -- write operation
                    for i in 7 downto 1 loop
                        RAM(i)<=RAM(i-1);
                    end loop;
                    RAM(0)<=di;
                    do <= di;
                else                            -- read operation
                    do <= RAM( conv_integer(addr));
                end if;
            end if;
          end if;
    end if;
  end process;


end Behavioral;
```

Στην RAM έχουμε ως είσοδο το ρολόι, την επίτρεψη εγγραφής we, το operation enable en, το reset την διεύθυνση του δεδομένου που θέλουμε, τα 8 bit που θέλουμε να γράψουμε σε μία θεση μνήμης της RAM και ως έξοδο μόνο τα δεδομένα εξόδου. Ορίζουμε επίσης ένα σήμα RAM που είναι ένας πίνακας από 8 8bit στοιχεία. Στην αρχή η RAM έχει 0 σε όλα της τα στοιχεία. Στην RAM χρησιμοποιούμε ένα process που στο sensititvity list έχει μόνο το ρολόι. Σε περίπτωση που το ρολόι έχει rising edge ελέγχουμε αρχικά αν το enable της RAM είναι 1. Αν το enable δεν είναι 1 τότε δεν κάνουμε τίποτα. Αν en='1' ελέγχουμε αν το reset είναι 1. Αν είναι τότε μηδενίζουμε όλα τα στοιχεία του πίνακα RAM και θέτουμε της έξοδο ίση με το πρώτο στοιχείο του πίνακα (πρακτικά θέτουμε την έξοδο ίση με 0). Αν το reset είναι 0 τότε ελέγχουμε αν το we (write enable) είναι 1. Σε περίπτωση που δεν είναι τότε θέτουμε την έξοδο ίση με το στοιχείο του πίνακα RAM με index που είναι ίσο με την δεκαδική αναπαράσταση της εισόδου addr. Σε περίπτωση όμως που το write enable είναι 1 τότε μετατοπίζουμε τα στοιχεία κατά «ένα πίσω» (δηλαδή RAM(7)=RAM(6), έπειτα RAM(6)=RAM(5) κτλ), θέτουμε το πρώτο στοιχείο του πίνακα RAM ίσο με την είσοδο di και τέλος θέτουμε την έξοδο do ίσο με το σήμα di.

Control Unit

Το control unit χρησιμεύει στο FIR ώστε να συγχρονίσουμε την RAM, την ROM, το MAC και για να παράγουμε την κατάλληλη χρονική στιγμή το '1' στο valid_out. Το control unit δηλαδή συγχρονίζει τις εξόδους της RAM και της ROM ώστε να πολλαπλασιάζονται τα κατάλληλα στοιχεία του h με το x, το πότε θα γίνει εγγραφή στην RAM, το πότε θα γίνει initialize του MAC και το πότε το valid_out θα γίνει 1. Ο κώδικας σε VHDL είναι:

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

entity control_unit is
  Port (
  clk, rst, valid_in: in std_logic;
  rom_address, ram_address: out std_logic_vector(2 downto 0);
  mac_init, valid_out: out std_logic;
  cycle8: out std_logic
    );
end control_unit;

architecture Behavioral of control_unit is
    component N_D_Flip_Flop is
        generic (
```

```
            N: integer :=7
        );
        Port (
            x, clk: in std_logic;
            y: out std_logic
        );
    end component;
    signal counter: unsigned(2 downto 0) := (others => '0');
    signal not_reset: std_logic :='1';
    signal prev_valid_in: std_logic :='0';
    signal valid_out_d: std_logic;
begin
    uut: N_D_Flip_Flop generic map(9) port map(valid_out_d, clk, valid_out);
    process(clk)
    begin
        if rising_edge(clk) then
            if rst='1' then
                counter<="010";
                rom_address<="111";
                mac_init<='1';
                valid_out_d<='0';
                not_reset<='0';
                cycle8<='0';
            else
                if counter=1 then
                    mac_init<='1';
                    cycle8<='0';
                    rom_address<="110";
                    ram_address<="001";
                    valid_out_d<='0';
                    counter<=counter+1;
                    not_reset<='1';
                else
                    mac_init<='0';
                    cycle8<='0';
                    valid_out_d<='0';
                    if prev_valid_in='1' and valid_in='0' then
                        prev_valid_in<='0';
                    end if;
                    if counter=0 and valid_in='1' and prev_valid_in<='0' then
                        cycle8<='1';
                        prev_valid_in<='1';
                        valid_out_d<='1';
                    end if;
                    ram_address<=std_logic_vector(counter);
                    rom_address<=std_logic_vector(7-counter);
                    if counter=7 then
                        counter<="000";
                    elsif counter=0 and valid_in='0' then
                        counter<="000";
                    else
                        counter<=counter+1;
                    end if;
                end if;
```

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

Για να επιτελεί το control unit όλα τα παραπάνω έχεις ως εισόδους το ρολόι, το reset, το valid_in και ως εξόδους το το mac_init για την αρχικοποίηση του MAC, το valid_out που δείχνει το αν είναι έγκυρη η έξοδος y του MAC, το cycle8 που χρησιμοποιείται ώστε να επιτρέψουμε την εγγραφή στην RAM καθώς και τα rom_address και ram_address που είναι οι διευθύνσεις που θέλουμε από την RAM και την ROM. Για να μπορέσουμε να συγχρονίσουμε κατάλληλα όλα τα παραπάνω σήματα θα χωρίσουμε την λειτουργία του control unit σε 8 κύκλους. Στον κύκλο 0 σε περίπτωση που το valid_in είναι 1 και προηγουμένως το valid_in ήταν 0 τότε θέτουμε το σήμα cycle8 σε 1, ώστε να γίνει εγγραφή στην RAM. Ο λόγος που ελέγχουμε αν το valid_in προηγουμένως ήταν 0 είναι για να σιγουρευτούμε ότι ο αποστολέας πραγματικά θέλει να στείλει αυτό το νέο δεδομένο στο FIR, καθώς σε περίπτωση που ο αποστολέας λειτουργεί σε χαμηλότερη συχνότητα από ότι το FIR τότε είναι πιθανό να περάσουν 8 κύκλοι του FIR και να μην έχει περάσει ένας κύκλος του αποστολέα για να κάνει 0 το valid in, με αποτέλεσμα το FIR να διαβάσει το ίδιο δεδομένο δύο φορές (ή και παραπάνω) . Επίσης στον κύκλο μηδέν θέτουμε το rom_address να ισούται με “111”και το ram_address να είναι “000”. Ωστόσο αν στον κύκλο 0 το valid in δεν είναι 1 τότε επαναλαμβάνουμε τον κύκλο 0, μέχρι να γίνει το valid_in ίσο με 1. Σε περίπτωση όμως που το valid_in είναι 1 τότε αυξάνουμε τον μετρητή των κύκλων κατά 1. Στον κύκλο 1 θέτουμε το σήμα mac_init ίσο με 1, θέτουμε το σήμα cycle8 ίσο με το 0, καθώς το είχαμε ορίσει στον προηγούμενο κύκλο ίσο με 1 , ζητάμε από την rom την διεύθυνση 6 και από την ram την διεύθυνση 1 και αυξάνουμε τον μετρητή των κύκλων. Στην συνέχεια για τον κύκλο 2 μέχρι τον κύκλο 6 μηδενίζουμε το mac_init θέτουμε την rom address ίση με 7-n και την ram address ίση με n, όπου n ο αριθμός του κύκλου κι αυξάνουμε τον μετρητή των κύκλων κατά 1. Τα ίδια γίνονται στον κύκλο 7 με την μόνη διαφορά ότι στο τέλος θέτουμε των μετρητή των κύκλων ίσο με το 0. Το μόνο σήμα που δεν έχουμε καλύψει μέχρι στιγμής είναι το valid_out. Για το valid_out χρησιμοποιούμε το component N_D_Flip_Flop που είναι μία αλληλουχία από D Flip Flop. Ο αριθμός των D Flip Flop δηλώνεται σε αυτό το component από το generic mapping. Η είσοδος αυτής της αλληλουχίας d flip flop είναι το σήμα valid_out_d και η έξοδος το σήμα valid_out. Το valid_out_d τίθεται 1 όταν επιτρέπουμε η εγγραφή στον κύκλο 0, ενώ στους άλλους κύκλους τίθεται 1.

FIR

Τέλος στο FIR component συνδέονται κατάλληλα τα components που έχουν προαναφερθεί μεταξύ τους. Ο VHDL κώδικας του FIR είναι ο ακόλουθος:

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

entity fir is
  Port (
        clk, rst, valid_in: in std_logic;
        x: in std_logic_vector(7 downto 0);
        y: out std_logic_vector(17 downto 0);
        valid_out: out std_logic
    );
end fir;

architecture Behavioral of fir is
    component control_unit is
      Port (
      clk, rst, valid_in: in std_logic;
      rom_address, ram_address: out std_logic_vector(2 downto 0);
      mac_init, valid_out: out std_logic;
      cycle8: 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
               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 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
                 reset: in std_logic;
              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 MAC is
        Port (
            x, h: in std_logic_vector(7 downto 0);
            mac_init: in std_logic;
            clk: in std_logic;
            y: out std_logic_vector(17 downto 0)
        );
    end component;
    signal rom_address, ram_address: std_logic_vector(2 downto 0):="000";
    signal mac_init, valid_out_temp, ram_enable, rom_enable: std_logic:='0';
    signal h, x_ram: std_logic_vector(7 downto 0):="00000000";
    signal cycle8: std_logic;
    signal write1: std_logic;
    signal x_mac, h_mac: std_logic_vector(7 downto 0);
    signal ram_address_out, rom_address_out: std_logic_vector(2 downto 0);
begin

    cu: control_unit port map(clk, rst, valid_in, rom_address, ram_address, mac_init,
valid_out, cycle8);

    rom: mlab_rom port map(clk, '1', rom_address, h);

    ram: mlab_ram port map(clk, cycle8, '1', rst, ram_address, x, x_ram);

    mac_fir: mac port map(x_ram, h, mac_init, clk, y);
end Behavioral;
```

Η σύνδεση μεταξύ τους είναι προφανής με βάση της εισόδους και της components του κάθε component. Το μόνο ίσως μη προφανές είναι ότι το ram_enable και το rom_enable είναι τίθεται πάντα στο 1 και ότι το reset πηγαίνει αυτούσιο στην ram και στον control unit.

Ζητούμενο 2ο: Testbench

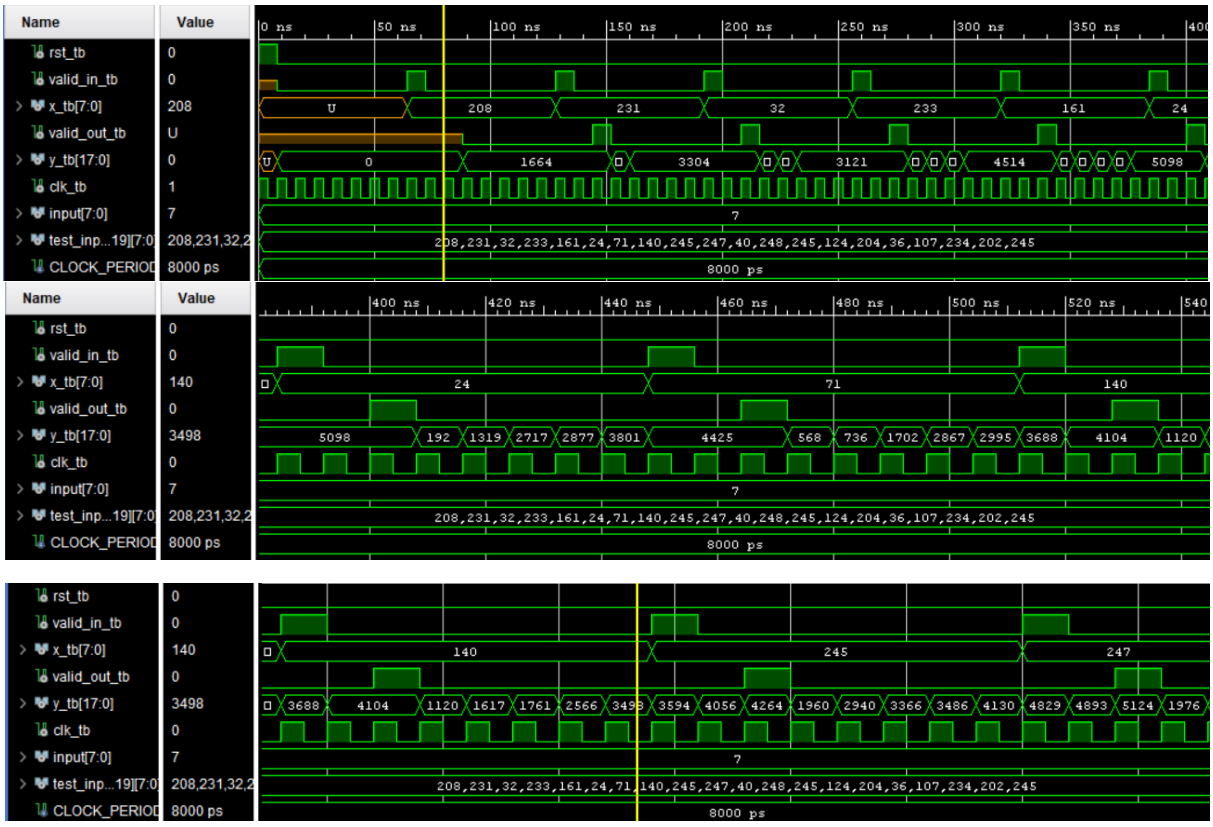
Σε αυτό το ζητούμενο θα κατασκευαστεί testbench με σκοπό να δείξουμε την σωστή λειτουργία του κυκλώματος. Το testbench είναι το ακόλουθο:

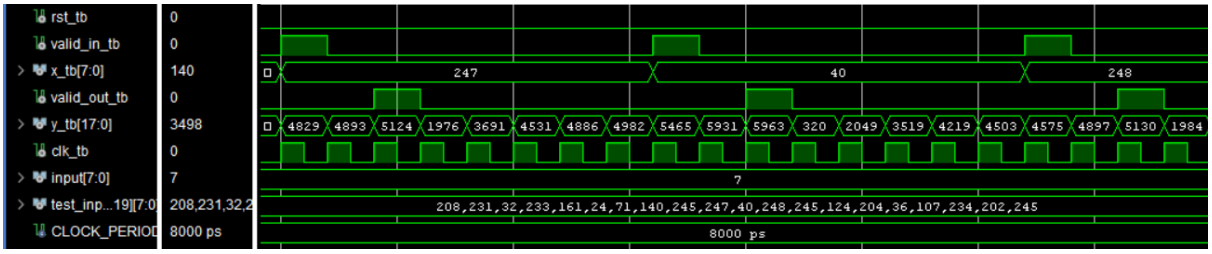
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity fir_testbench is
end fir_testbench;

architecture Behavioral of fir_testbench is
    component fir is
      Port (
            clk, rst, valid_in: in std_logic;
            x: in std_logic_vector(7 downto 0);
            y: out std_logic_vector(17 downto 0);
            valid_out: out std_logic
        );
    end component;
    signal rst_tb, valid_in_tb: std_logic;
    signal x_tb: std_logic_vector(7 downto 0);
    signal valid_out_tb: std_logic;
    signal y_tb: std_logic_vector(17 downto 0);
    signal clk_tb:std_logic;
    constant CLOCK_PERIOD: time := 8 ns;
    signal input: unsigned(7 downto 0):="00000111";
    type inputs is array (0 to 19) of std_logic_vector (7 downto 0);
    signal test_inputs : inputs :=("11010000", "11100111", "00100000", "11101001",
"10100001", "00011000", "01000111", "10001100", "11110101", "11110111", "00101000",
"11111000", "11110101", "01111100", "11001100", "00100100", "01101011", "11101010",
"11001010", "11110101");
begin

uut: fir port map(clk_tb, rst_tb, valid_in_tb, x_tb, y_tb, valid_out_tb);

 clk_process: process
   begin
      clk_tb <= '1';
      wait for CLOCK_PERIOD / 2;
      clk_tb <= '0';
      wait for CLOCK_PERIOD / 2;
  end process;

  stimulus_process: process
  begin
      rst_tb<='1';
      wait for CLOCK_PERIOD;
      rst_tb<='0';
      valid_in_tb<='0';
      wait for 7*CLOCK_PERIOD;
```

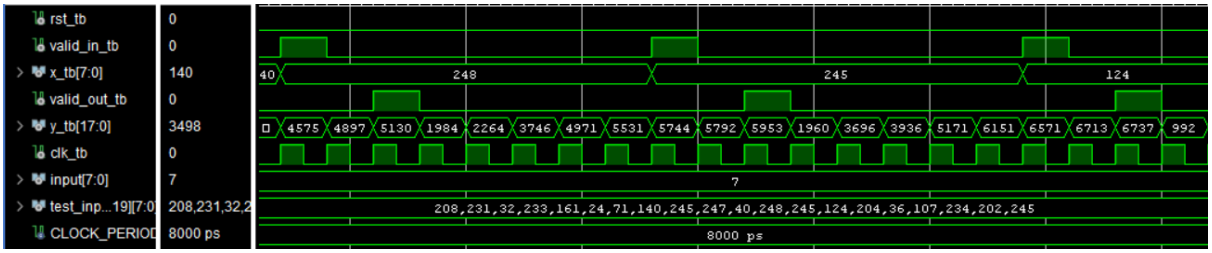
```
      for i in 0 to 19 loop

                        valid_in_tb<='1';
                        x_tb<=std_logic_vector(test_inputs(i));
                        wait for CLOCK_PERIOD;
                        valid_in_tb<='0';
                        wait for 7*CLOCK_PERIOD;
            end loop;
      valid_in_tb<='0';
      wait for 7*CLOCK_PERIOD;
      rst_tb<='1';
      wait for CLOCK_PERIOD;
      rst_tb<='0';
      wait for 20*CLOCK_PERIOD;

 end process;

end Behavioral;
```

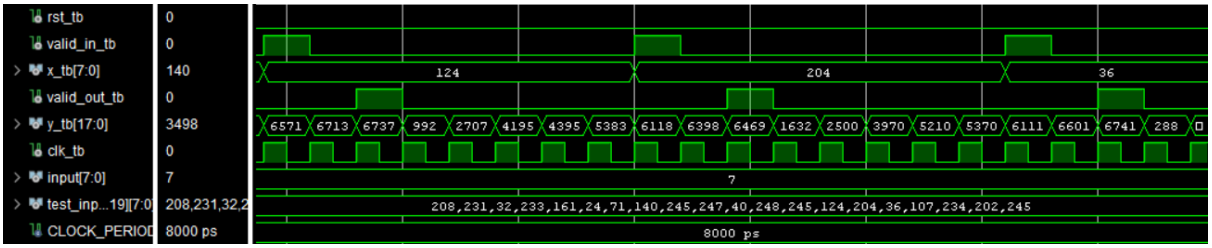
Σε αυτό το testbench ελέγχουμε στην αρχή το reset και στην συνέχεια ελέγχουμε την έξοδο του κυκλώματος με βάση το input που υπάρχει στον πίνακα (η είσοδος είναι ίδια με αυτή που μας ζητήθηκε στο εργαστήριο) και να δίνεται την σωστή χρονική στιγμή το valid_out. Οι διάφορες είσοδοι δίνονται με διαφορά 8 κύκλων ρολογιού. Μόλις τελειώσει το input ελέγχουμε πάλι την λειτουργία του reset και ελέγχουμε αν το κύκλωμα θα πάρει το τελευταίο input ενώ το valid_in είναι 0.

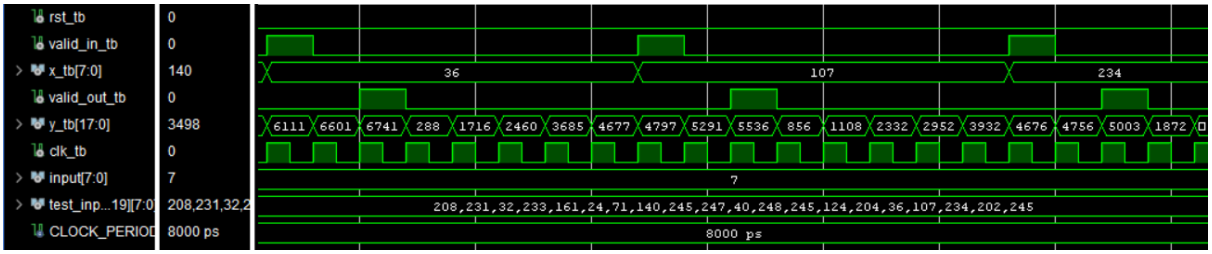
Το αποτέλεσμα της προσομοίωσης είναι το ακόλουθο:

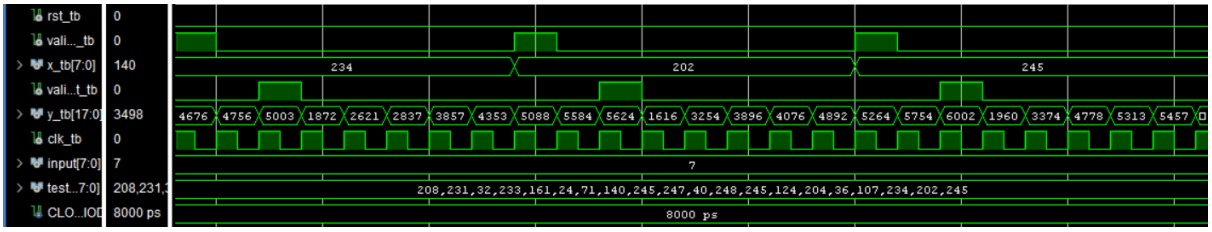


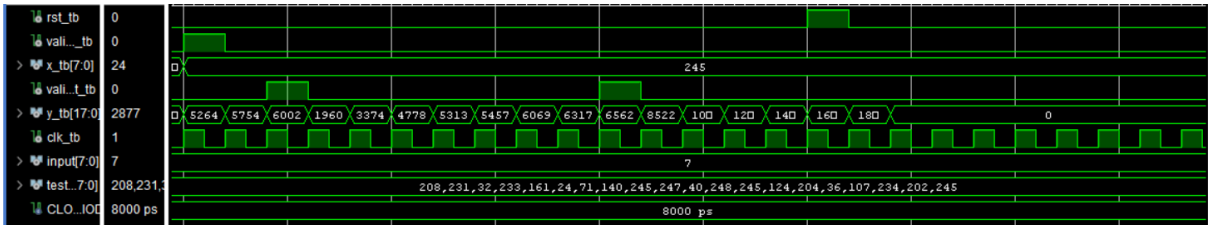












Βλέπουμε λοιπόν ότι το FIR λειτουργεί όπως αναμενόταν.

Ζητούμενο 3ο

Αρχικά θα ελέγξουμε το κρίσιμο μονοπάτι του FIR. Κάνοντας synthesis και επιλέγοντας timing summary βλέπουμε τα ακόλουθα:

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	9	9	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[17]/D	5.547	2.483	3.064	∞
Path 2	∞	9	9	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[16]/D	5.434	2.370	3.064	∞
Path 3	∞	8	8	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[13]/D	5.433	2.369	3.064	∞
Path 4	∞	8	8	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[15]/D	5.414	2.350	3.064	∞
Path 5	∞	8	8	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[14]/D	5.341	2.277	3.064	∞
Path 6	∞	8	8	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[12]/D	5.320	2.256	3.064	∞
Path 7	∞	7	7	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[9]/D	5.309	2.545	2.764	∞
Path 8	∞	7	7	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[11]/D	5.290	2.526	2.764	∞
Path 9	∞	7	7	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[10]/D	5.217	2.453	2.764	∞
Path 10	∞	7	7	20	rom/rom_out_reg[1]/C	mac_fir/sum_reg[8]/D	5.196	2.432	2.764	∞

Βλέπουμε λοιπόν ότι το κρίσιμο μονοπάτι είναι από την ROM στο τελευταίο ψηφίο του sum στο MAC και η καθυστέρησή του είναι 5,547ns. Έτσι η συχνότητα λειτουργίας του FPGA είναι 180.28MHz.

Οι πόροι που χρησιμοποιήθηκαν για την υλοποίηση του φίλτρου είναι φαίνονται παρακάτω:

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	91	17600	0.52
LUTRAM	1	6000	0.02
FF	126	35200	0.36
IO	30	100	30.00
BUFG	1	32	3.13