ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΙΣΤΩΝ

ΨΗΦΙΑΚΑ VLSI – 8ο ΕΞΑΜΗΝΟ

Μαρκέτος Νικόδημος – ΑΜ: 03117095

Τζομάκα Αφροδίτη – ΑΜ: 03117107

Ομάδα Β2

Ζήτημα 1: FIR Filter

Στο ζητούμενο αυτό κληθήκαμε να υλοποιήσουμε της βασικές μονάδες του σχήματος της εκφώνησης προκειμένου να συνθέσουμε ένα FIR φίλτρο. Ακολουθούν οι αντίστοιχοι κώδικες:

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;
```

RAM

```
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;
          rst  : in std_logic;
          we   : in std_logic;                      --- memory write enable
          en   : in std_logic;                      --- operation enable
          addr : in std_logic_vector(2 downto 0);           -- memory address
          di   : in std_logic_vector(data_width-1 downto 0);        -- input data
          do   : out std_logic_vector(data_width-1 downto 0));      -- output data
end mlab_ram;

architecture Behavioral of mlab_ram is

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

begin
    process (clk,rst)
    begin
        if rst = '1' then
                RAM <= (others => (others => '0'));
        else
            if  clk'event and clk = '1' then
                 if en = '1' then
                    if we = '1' then
                        RAM(7 downto 1 ) <= ram(6 downto 0);                -- write operation
                        RAM(conv_integer(addr)) <= di;
                        do <= di;
                 else
                        do <= RAM( conv_integer(addr));
                    end if;
                end if;
            end if;
       end if;
    end process;

end Behavioral;
```

MAC

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.all;

entity MAC is
    generic(
        data_width :   integer    := 8;
        coeff_width:   integer    := 8;
        min_vector_size: integer  := 8
    );
    Port (
        rom_out: in std_logic_vector(data_width-1 downto 0);
        ram_out: in std_logic_vector(coeff_width-1 downto 0);
        mac_init : in std_logic;
```

```
        L: out std_logic_vector((data_width + coeff_width + integer(ceil(log2(real(min_vector_size)))))-
1  downto 0);
        clk : in std_logic
    );
end MAC;

architecture Behavioral of MAC is
        signal temp : std_logic_vector((data_width + coeff_width + integer(ceil(log2(real(min_vector_size))))
) -1 downto 0):= (others => '0') ;
begin
        process(clk)
            begin
                 if (rising_edge(clk)) then
                        if mac_init = '1' then
                            temp <= (others => '0');
                              temp(data_width + coeff_width-1 downto 0) <= ram_out * rom_out;
                        else
                            temp <= temp + (ram_out * rom_out);
                        end if;
                        L <= temp;
                 end if;
        end process;
end Behavioral;
```

Control Unit

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


entity ControlUnit is
    Port (
            clk : in std_logic;
            valid_in : in std_logic;
            rom_address : out std_logic_vector(2 downto 0);
            mac_init : out std_logic;
            ram_address : out std_logic_vector(2 downto 0);
            we : out std_logic;
            valid_out: out std_logic
         );
end ControlUnit;

architecture Behavioral of ControlUnit is
    signal counter : std_logic_vector(2 downto 0):= (others => '0');
    signal temp_valid_out :std_logic_vector(9 downto 0 ):= (others => '0');
begin
    process(clk)
        begin
        if(rising_edge(clk)) then
            if counter  = "000" then
                we<= valid_in;
                mac_init<= '1';
                temp_valid_out(0) <= valid_in;
            else
                temp_valid_out(0) <= valid_in;
                we<= valid_in;
                mac_init <= '0';
            end if;
            ram_address <= counter;
            rom_address <= counter;
            counter <= counter + 1;
            valid_out <= temp_valid_out(9);
        end if;
    end process;
```

```
    process(clk)
        begin
            if (rising_edge(clk)) then
                    temp_valid_out(9 downto 1)<= temp_valid_out(8 downto 0);
            end if;
        end process;

end Behavioral;
```

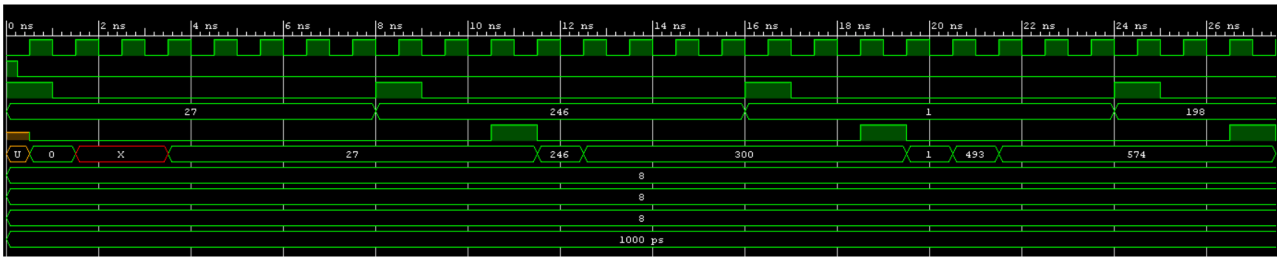
Testbench

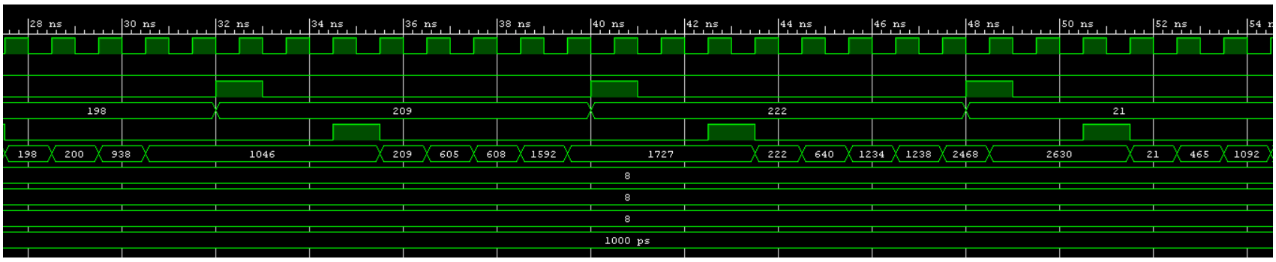
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.all;

entity FIR_TB is
end FIR_TB;

architecture testbench of FIR_TB is
    constant data_width, coeff_width : integer := 8;
    constant min_vector_size : integer := 8;

    component FIR
        generic(
            data_width :   integer    := 8;
            coeff_width:   integer    := 8;
            min_vector_size: integer  := 8
        );
        Port (
            clk: in std_logic;
            rst: in std_logic;
            valid_in : in std_logic;
            x: in std_logic_vector(data_width-1 downto 0);
            y: out std_logic_vector((data_width + coeff_width + integer(ceil(log2(real(min_vector_size)))))-
1  downto 0);
            valid_out : out std_logic
        );
    end component;

    --Input Signals
    signal clk: std_logic:='0';
    signal rst: std_logic:='0';
    signal valid_in: std_logic:='0';
    signal x : std_logic_vector(data_width-1 downto 0):=(others=>'0');
    signal valid_out: std_logic:='0';

    --output signals
    signal y : std_logic_vector((data_width + coeff_width + integer(ceil(log2(real(min_vector_size)))))-
1 downto 0):=(others=>'0');

    --Clock
    constant CLKP : time := 1ns;

    begin
        clk_proc:
            process
                begin
                    clk <= '0';
                    wait for CLKP/2;
                    clk <= '1';
                    wait for CLKP/2;
            end process;
```

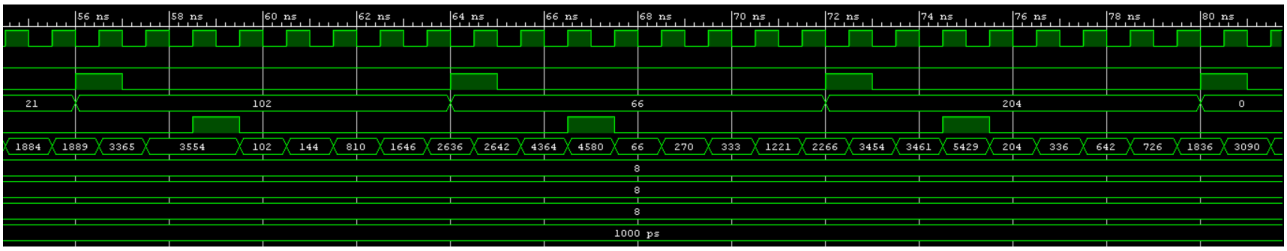
```
        UUT: FIR port map(
            clk => clk,
            rst => rst,
            valid_in => valid_in,
            x => x ,
            y => y,
            valid_out => valid_out
            );
        TEST:
            process
            begin
                rst <= '1';
                valid_in <= '1';
                x <= "00011011";
                wait for CLKP/4;
                rst<='0';
                wait for 3*CLKP/4;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "11110110";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "00000001";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "11000110";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "11010001";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "11011110";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "00010101";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "01100110";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "01000010";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "11001100";
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                x <= "00000000";
                wait for CLKP;
```

```
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
                valid_in <= '1';
                wait for CLKP;
                valid_in <= '0';
                wait for 7*CLKP;
            end process;
end testbench;
```

Οι τιμές για το testbench είναι εκείνες που δόθηκαν κατά την εξέταση και ακολουθεί η επίδειξη της ορθής λειτουργίας:







RESOURCE UTILIZATION:

Resource	Utilization	Available	Utilization %
LUT	139	17600	0.79
LUTRAM	1	6000	0.02
FF	132	35200	0.38
IO	31	100	31.00

TIMING SUMMARY:

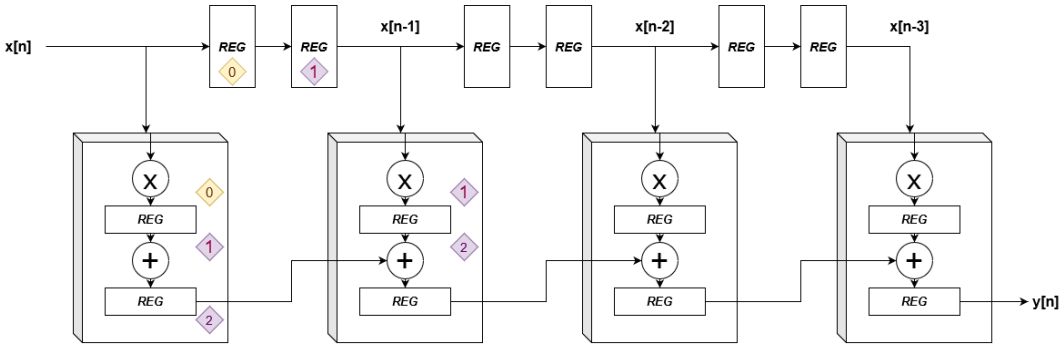
Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	8	8	20	ROM/rom_out_reg[1]/C	MC/temp_reg[11]/D	6.215	2.833	3.382
Path 2	∞	8	8	20	ROM/rom_out_reg[1]/C	MC/temp_reg[10]/D	5.970	2.754	3.216
Path 3	∞	8	8	20	ROM/rom_out_reg[1]/C	MC/temp_reg[9]/D	5.921	2.845	3.076
Path 4	∞	8	8	20	ROM/rom_out_reg[1]/C	MC/temp_reg[8]/D	5.802	2.727	3.075
Path 5	∞	7	7	20	ROM/rom_out_reg[1]/C	MC/temp_reg[7]/D	5.727	2.345	3.382
Path 6	∞	9	9	20	ROM/rom_out_reg[1]/C	MC/temp_reg[17]/D	5.547	2.483	3.064
Path 7	∞	9	9	20	ROM/rom_out_reg[1]/C	MC/temp_reg[18]/D	5.455	2.391	3.064
Path 8	∞	9	9	20	ROM/rom_out_reg[1]/C	MC/temp_reg[16]/D	5.434	2.370	3.064
Path 9	∞	8	8	20	ROM/rom_out_reg[1]/C	MC/temp_reg[13]/D	5.433	2.369	3.064
Path 10	∞	8	8	20	ROM/rom_out_reg[1]/C	MC/temp_reg[15]/D	5.414	2.350	3.064

Άρα το critical path εντοπίζεται από την ROM στην MAC με συνολική καθυστέρηση 6.215ns.

Στο τέλος της παρουσίασης των τριών ζητουμένων θα γίνει η σύγκριση των αποτελεσμάτων για το resource utilization και το critical path κάθε υλοποίησης.

Ζήτημα 2: FIR Pipelined

Στο συγκεκριμένο θέμα, καλούμαστε να «ξεδιπλώσουμε» το φίλτρο που υλοποιήσαμε στο παραπάνω ζήτημα μετατρέποντάς το σε πλήρως pipelined. Η λογική που ακολουθήθηκε φαίνεται στο παρακάτω σχήμα.



Κάθε κουτί του παραπάνω σχήματος αντιστοιχεί σε ένα MAC Unit το οποίο είναι υπεύθυνο για τον πολλαπλασιασμό και την πρόσθεση που απαιτεί η πράξη της συνέλιξης. Χρησιμοποιήσαμε και εντός αυτού registers ώστε η υλοποίηση μας να είναι πλήρως pipelined και να μειώσουμε το critical path. Το σχήμα δείχνει την ροή δεδομένων για ένα 4 tap φίλτρο λόγω απλότητας, ωστόσο η ίδια λογική εύκολα επεκτείνεται και για το δικό μας 8 tap φίλτρο. Ποιο συγκεκριμένα, εκείνο που θέλουμε για να έχουμε σωστό αποτέλεσμα είναι τα δεδομένα από τον αθροιστή του τωρινού MAC Unit να είναι

διαθέσιμα την ίδια χρονική στιγμή με τα δεδομένα από τον πολλαπλασιαστή του επόμενου MAC Unit. Προκειμένου να επιτευχθεί αυτό χρειαζόμαστε δύο καθυστερήσεις και άρα δύο registers. Το παραπάνω σχήμα περιέχει και ένα παράδειγμα του χρονισμού όπου φαίνεται πραγματικά ότι με αυτήν την σχεδίαση έχουμε τα δύο ζητούμενα αποτελέσματα στον ίδιο κύκλο (τον κύκλο 2 στο παράδειγμα). Το αρχικό $T_{latency}$ ισούται με 8.

Για την υλοποίηση αυτή λοιπόν έχουμε τον παρακάτω κώδικα, μαζί με το αντίστοιχο testbench:

MAC Unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.math_real.all;

entity MAC_pipelined is
        generic (
                M : integer;
                data_width : integer;
                coeff_width : integer
                );
        Port (
            clk : in std_logic;
            rst : in std_logic;
            valid_in : in std_logic;
            x : in std_logic_vector (data_width-1 downto 0);
            coeff : in std_logic_vector (coeff_width-1 downto 0);
            partial_sum : in std_logic_vector (data_width+coeff_width+integer(ceil(log2(real(M))))-
1 downto 0);
            mac_out : out std_logic_vector (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0)
            );
end MAC_pipelined;

architecture Behavioral of MAC_pipelined is

    signal tmp_mul : std_logic_vector (data_width+coeff_width-1 downto 0);
 -- output of multiplication
    signal tmp_sum : std_logic_vector (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
 -- output of addition
    signal mul_reg : std_logic_vector (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
 -- reg of multiplication
    signal sum_reg : std_logic_vector (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
 -- reg of addition

begin
    tmp_mul <= x * coeff;
    tmp_sum <= mul_reg + partial_sum;
    mac_out <= sum_reg;

    process(rst,clk)
        begin
        if rst = '1' then
            mul_reg <= (others => '0');
            sum_reg <= (others => '0');
        elsif rising_edge(clk) then
            if valid_in = '1' then
                mul_reg(data_width+coeff_width-1 downto 0) <= tmp_mul;
                mul_reg(data_width+coeff_width+integer(ceil(log2(real(M))))-
1 downto data_width+coeff_width) <= (others => '0');
                sum_reg <= tmp_sum;
            end if;
        end if;
    end process;
end Behavioral;
```

Code of FIR_pipelined

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.math_real.all;

entity FIR_pipelined is
        generic (
                M : integer := 8;
                data_width : integer := 8;
                coeff_width : integer := 8
                );
        Port (
                clk : in std_logic;
                rst : in std_logic;
                valid_in :in std_logic;
                x : in std_logic_vector(data_width-1 downto 0);
                valid_out : out std_logic;
                fir_out : out std_logic_vector
                (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0)
            );
end FIR_pipelined;

architecture Behavioral of FIR_pipelined is
    component MAC_pipelined is
            generic (
                    M : integer;
                    data_width : integer;
                    coeff_width : integer
                    );
            Port (
                    clk : in std_logic;
                    rst : in std_logic;
                    valid_in : in std_logic;
                    x : in std_logic_vector (data_width-1 downto 0);
                    coeff : in std_logic_vector (coeff_width-1 downto 0);
                    partial_sum : in std_logic_vector
                    (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
                    mac_out : out std_logic_vector
                    (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0)
                );
    end component;

    signal reg11, reg21, reg31, reg41, reg51, reg61, reg71 : std_logic_vector (data_width downto 0);
    signal reg12, reg22, reg32, reg42, reg52, reg62, reg72 : std_logic_vector (data_width-1 downto 0);
    signal tmp_sum0, tmp_sum1, tmp_sum2, tmp_sum3, tmp_sum4, tmp_sum5, tmp_sum6:std_logic_vector
           (data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
    signal valid8, valid9: std_logic;

begin
    CELL0:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (  clk => clk, rst => rst, valid_in => valid_in,
                    x => x, coeff => "00000001", partial_sum => (others => '0'), mac_out => tmp_sum0
                 );

    CELL1:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (  clk => clk, rst => rst, valid_in => valid_in,
                    x => reg12, coeff => "00000010", partial_sum => tmp_sum0, mac_out => tmp_sum1
                 );

    CELL2:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (  clk => clk, rst => rst, valid_in => valid_in,
                    x => reg22, coeff => "00000011", partial_sum => tmp_sum1, mac_out => tmp_sum2
```

```
            );

  CELL3:MAC_pipelined
      generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
      port map (  clk => clk, rst => rst, valid_in => valid_in,
                  x => reg32, coeff => "00000100", partial_sum => tmp_sum2, mac_out => tmp_sum3
            );

  CELL4:MAC_pipelined
      generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
      port map (  clk => clk, rst => rst, valid_in => valid_in,
                  x => reg42, coeff => "00000101", partial_sum => tmp_sum3, mac_out => tmp_sum4
            );

  CELL5:MAC_pipelined
      generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
      port map (  clk => clk, rst => rst, valid_in => valid_in,
                  x => reg52, coeff => "00000110", partial_sum => tmp_sum4, mac_out => tmp_sum5
            );

  CELL6:MAC_pipelined
      generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
      port map (  clk => clk, rst => rst, valid_in => valid_in,
                  x => reg62, coeff => "00000111", partial_sum => tmp_sum5, mac_out => tmp_sum6
            );

  CELL7:MAC_pipelined
      generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
      port map (  clk => clk, rst => rst, valid_in => valid_in,
                  x => reg72, coeff => "00001000", partial_sum => tmp_sum6, mac_out => fir_out
            );

  valid_out <= valid9;

process(clk,rst)
begin
  if rst = '1' then
    reg11 <= (others => '0'); reg21 <= (others => '0'); reg31 <= (others => '0'); reg41 <= (others => '0');
    reg51 <= (others => '0'); reg61 <= (others => '0'); reg71 <= (others => '0');

    reg12 <= (others => '0'); reg22 <= (others => '0'); reg32 <= (others => '0'); reg42 <= (others => '0');
    reg52 <= (others => '0'); reg62 <= (others => '0'); reg72 <= (others => '0');

    valid8 <= '0'; valid9 <= '0';

  elsif rising_edge(clk) then

      if valid_in='1' then
           valid9 <= valid8;
           valid8 <= reg71(8);
           reg72 <= reg71(7 downto 0);
           reg71(7 downto 0) <= reg62;
           reg71(8) <= reg61(8);
           reg62 <= reg61(7 downto 0);
           reg61(7 downto 0) <= reg52;
           reg61(8) <= reg51(8);
           reg52 <= reg51(7 downto 0);
           reg51(7 downto 0) <= reg42;
           reg51(8) <= reg41(8);
           reg42 <= reg41(7 downto 0);
           reg41(7 downto 0) <= reg32;
           reg41(8) <= reg31(8);
           reg32 <= reg31(7 downto 0);
           reg31(7 downto 0) <= reg22;
           reg31(8) <= reg21(8);
           reg22 <= reg21(7 downto 0);
           reg21(7 downto 0) <= reg12;
```

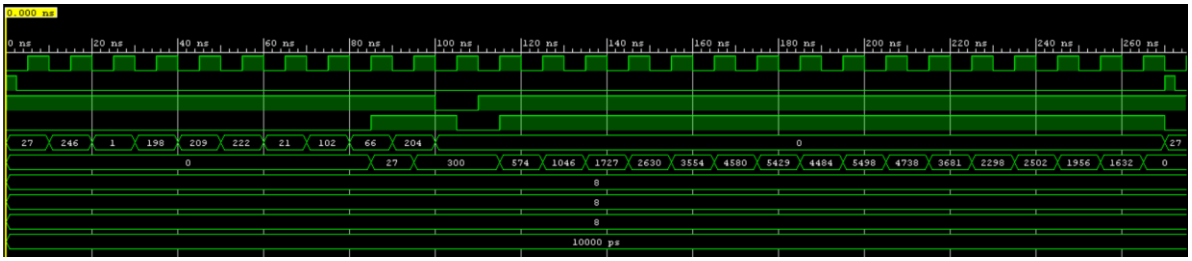
```
            reg21(8) <= reg11(8);
            reg12 <= reg11(7 downto 0);
            reg11(7 downto 0) <= x;
            reg11(8) <= valid_in;
         else
            valid9 <= '0';
         end if;
      end if;
   end process;

end Behavioral;
```

Code of FIR_pipelined_tb Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.math_real.all;

entity FIR_pipelined_tb is
end FIR_pipelined_tb;

architecture Behavioral of FIR_pipelined_tb is
    constant M, data_width, coeff_width : integer := 8;
    component FIR_pipelined is
        generic (
                M : integer := 8;
                data_width : integer := 8;
                coeff_width : integer := 8
                );
        Port (
            clk : in std_logic;
            rst : in std_logic;
            valid_in :in std_logic;
            x : in std_logic_vector (data_width-1 downto 0);
            valid_out : out std_logic;
            fir_out : out std_logic_vector(data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0)
            );
    end component;

    signal clk, rst, valid_in, valid_out: std_logic;
    signal x : std_logic_vector(data_width-1 downto 0);
    signal fir_out : std_logic_vector(data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
    constant CLKP : time := 10ns;
    begin
        UUT: FIR_pipelined
            port map(
                    clk => clk,
                    rst => rst,
                    valid_in => valid_in,
                    x => x,
                    valid_out => valid_out,
                    fir_out => fir_out
                    );
        clk_proc:process
            begin
                clk <= '0';
                wait for CLKP/2;
                clk <= '1';
                wait for CLKP/2;
        end process clk_proc;
        test:process
            begin
                rst <= '1';
                valid_in <= '1';
                x <= "00011011";
```

```
                wait for CLKP/4;
                rst<='0';
                wait for 3*CLKP/4;
                valid_in <= '1';
                x <= "11110110";
                wait for CLKP;
                x <= "00000001";
                wait for CLKP;
                x <= "11000110";
                wait for CLKP;
                x <= "11010001";
                wait for CLKP;
                x <= "11011110";
                wait for CLKP;
                x <= "00010101";
                wait for CLKP;
                x <= "01100110";
                wait for CLKP;
                x <= "01000010";
                wait for CLKP;
                x <= "11001100";
                wait for CLKP;
                x <= (others => '0');
                valid_in <= '0';
                wait for CLKP;
                valid_in <= '1';
                wait for 16*CLKP;
        end process;
end Behavioral;
```

Οι τιμές για το testbench είναι εκείνες που δόθηκαν κατά την εξέταση και ακολουθεί η επίδειξη της ορθής λειτουργίας:



RESOURCE UTILIZATION:

Resource	Utilization	Available	Utilization %
LUT	106	17600	0.60
LUTRAM	1	6000	0.02
FF	261	35200	0.74
IO	31	100	31.00

TIMING REPORT:

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	2	1	CELL7/sum_reg_reg[0]/C	fir_out[0]	4.076	3.276	0.800
Path 2	∞	2	2	1	CELL7/sum_reg_reg[10]/C	fir_out[10]	4.076	3.276	0.800
Path 3	∞	2	2	1	CELL7/sum_reg_reg[11]/C	fir_out[11]	4.076	3.276	0.800
Path 4	∞	2	2	1	CELL7/sum_reg_reg[12]/C	fir_out[12]	4.076	3.276	0.800
Path 5	∞	2	2	1	CELL7/sum_reg_reg[13]/C	fir_out[13]	4.076	3.276	0.800
Path 6	∞	2	2	1	CELL7/sum_reg_reg[14]/C	fir_out[14]	4.076	3.276	0.800
Path 7	∞	2	2	1	CELL7/sum_reg_reg[15]/C	fir_out[15]	4.076	3.276	0.800
Path 8	∞	2	2	1	CELL7/sum_reg_reg[1]/C	fir_out[1]	4.076	3.276	0.800
Path 9	∞	2	2	1	CELL7/sum_reg_reg[2]/C	fir_out[2]	4.076	3.276	0.800
Path 10	∞	2	2	1	CELL7/sum_reg_reg[3]/C	fir_out[3]	4.076	3.276	0.800

Το critical path σε αυτήν την υλοποίηση είναι ίσο με 4.076ns.

Ζήτημα 3: FIR Parallel

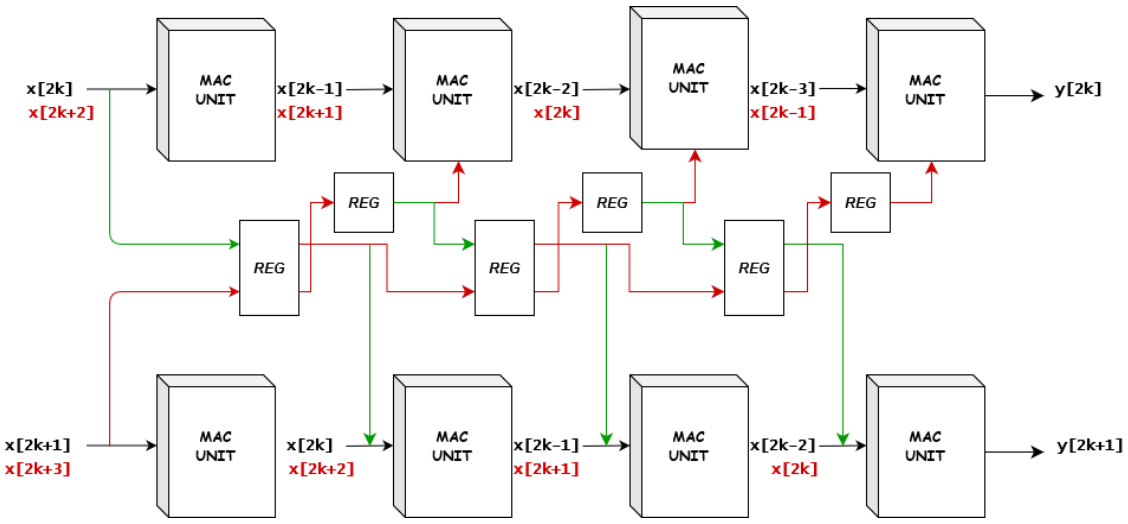
Στο ζητούμενο αυτό προσθέτουμε στην υλοποίηση μας ένα βαθμό παραλληλίας επομένως έχουμε δύο ταυτόχρονες εισόδους ζητώντας και δύο ταυτόχρονες εξόδους. Ουσιαστικά, μελετώντας τις εξισώσεις που δίνονται και στην εκφώνηση

$$y[2k] = x[2k]*h[0] + x[2k-1]*h[1] + x[2k-2]*h[2] + x[2k-3]*h[3]$$

$$y[2k+1] = x[2k+1]*h[0] + x[2k+1-1]*h[1] + x[2k+1-2]*h[2] + x[2k+1-3]*h[3]$$

παρατηρούμε ότι ουσιαστικά θα χρειαστούμε την ίδια σχεδόν υποδομή δύο φορές.

Το σχήμα δείχνει την ροή δεδομένων για ένα 4 tap φίλτρο λόγω απλότητας, ωστόσο η ίδια λογική εύκολα επεκτείνεται και για το δικό μας 8 tap φίλτρο. Πιο συγκεκριμένα, βλέπουμε στο παρακάτω διάγραμμα δύο διαδοχικούς κύκλους ρολογιού, ο πρώτος με **μαύρο** και ο δεύτερος με **κόκκινο**. Παρατηρούμε ότι πχ το ***χ[2k]*** θα χρειαστεί στον ίδιο κύκλο και στην «κάτω» γραμμή υπολογισμού του ***χ[2k+1],*** επομένως για να συγχρονιστεί με τον εκάστοτε πολλαπλασιασμό θα χρειαστεί μόνο ένα επίπεδο καθυστέρησης. Αντίστοιχα, το ***χ[2k+1]*** θα χρειαστεί για τον υπολογισμό του ***χ[2k+2],*** στον επόμενο κύκλο, στην «πάνω» γραμμή υπολογισμού, επομένως θα χρειαστεί δύο επίπεδα καθυστέρησης.



Σύμφωνα λοιπόν με τις παραπάνω παρατηρήσεις και διάγραμμα αναπτύσσουμε την υλοποίησή μας σύμφωνα με τους κώδικες που ακολουθούν:

*Η MAC που χρησιμοποιήθηκε είναι η ίδια με του προηγούμενου ζητουμένου,

Code of FIR_pipelined_tb Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.math_real.all;

entity FIR_parallel is
        generic (
                M : integer := 8;
```

```
                data_width : integer := 8;
                coeff_width : integer := 8
                );
        Port (
                clk : in std_logic;
                rst : in std_logic;
                valid_in :in std_logic;
                x1 : in std_logic_vector(data_width-1 downto 0);
                x2 : in std_logic_vector(data_width-1 downto 0);
                valid_out : out std_logic;
                fir_out1 : out std_logic_vector(data_width+coeff_width+integer(ceil(log2(real(M))))-
1 downto 0);
                fir_out2 : out std_logic_vector(data_width+coeff_width+integer(ceil(log2(real(M))))-
1 downto 0)
            );
end FIR_parallel;

architecture Behavioral of FIR_parallel is
    component MAC_pipelined is
            generic (
                    M : integer;
                    data_width : integer;
                    coeff_width : integer
                    );
            Port (
                  clk : in std_logic;
                  rst : in std_logic;
                  valid_in : in std_logic;
                  x : in std_logic_vector (data_width-1 downto 0);
                  coeff : in std_logic_vector (coeff_width-1 downto 0);
                  partial_sum : in std_logic_vector (data_width+coeff_width+integer(ceil(log2(real(M))))-
1 downto 0);
                  mac_out : out std_logic_vector (data_width+coeff_width+integer(ceil(log2(real(M))))-
1 downto 0)
                );
    end component;

    signal reg11, reg21, reg31, reg41, reg51, reg61, reg71 : std_logic_vector (2*data_width downto 0);
    signal reg12, reg22, reg32, reg42, reg52, reg62, reg72 : std_logic_vector (data_width-1 downto 0);
    signal  tmp_sum10, tmp_sum11, tmp_sum12, tmp_sum13, tmp_sum14, tmp_sum15, tmp_sum16,
            tmp_sum20, tmp_sum21, tmp_sum22, tmp_sum23, tmp_sum24, tmp_sum25, tmp_sum26:std_logic_vector(data
_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
    signal valid8, valid9: std_logic;

begin
    CELL10:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => x1, coeff => "00000001", partial_sum => (others => '0'), mac_out => tmp_sum10
                 );
    CELL20:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => x2, coeff => "00000001", partial_sum => (others => '0'), mac_out => tmp_sum20
                );
-------------------------------------------------------------------------------------------------
    CELL11:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg12, coeff => "00000010", partial_sum => tmp_sum10, mac_out => tmp_sum11
                 );
    CELL21:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg11(7 downto 0), coeff => "00000010", partial_sum => tmp_sum20, mac_out => tmp_sum21
                );
-------------------------------------------------------------------------------------------------
```

```
    CELL12:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg22, coeff => "00000011", partial_sum => tmp_sum11, mac_out => tmp_sum12
                 );
    CELL22:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg21(7 downto 0), coeff => "00000011", partial_sum => tmp_sum21, mac_out => tmp_sum22
                );
-------------------------------------------------------------------------------------------------------
    CELL13:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg32, coeff => "00000100", partial_sum => tmp_sum12, mac_out => tmp_sum13
                 );
    CELL23:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg31(7 downto 0), coeff => "00000100", partial_sum => tmp_sum22, mac_out => tmp_sum23
                );
-------------------------------------------------------------------------------------------------------
    CELL14:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg42, coeff => "00000101", partial_sum => tmp_sum13, mac_out => tmp_sum14
                 );
    CELL24:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg41(7 downto 0), coeff => "00000101", partial_sum => tmp_sum23, mac_out => tmp_sum24
                );
-------------------------------------------------------------------------------------------------------
    CELL15:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg52, coeff => "00000110", partial_sum => tmp_sum14, mac_out => tmp_sum15
                 );
    CELL25:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg51(7 downto 0), coeff => "00000110", partial_sum => tmp_sum24, mac_out => tmp_sum25
                );
-------------------------------------------------------------------------------------------------------

    CELL16:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (  clk => clk, rst => rst, valid_in => valid_in,
                    x => reg62, coeff => "00000111", partial_sum => tmp_sum15, mac_out => tmp_sum16
                 );
    CELL26:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg61(7 downto 0), coeff => "00000111", partial_sum => tmp_sum25, mac_out => tmp_sum26
                );
-------------------------------------------------------------------------------------------------------

    CELL17:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg72, coeff => "00001000", partial_sum => tmp_sum16, mac_out => fir_out1
                 );
    CELL27:MAC_pipelined
        generic map (M => M,    data_width => data_width,   coeff_width => coeff_width)
        port map (clk => clk, rst => rst, valid_in => valid_in,
                  x => reg71(7 downto 0), coeff => "00001000", partial_sum => tmp_sum26, mac_out => fir_out2
                );
```

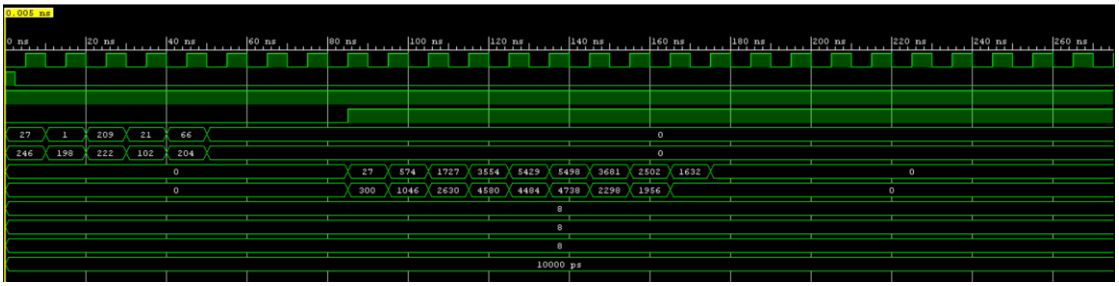
```
----------------------------------------------------------------------------------------------------
    valid_out <= valid9;

    process(clk,rst)
    begin
        if rst = '1' then
            reg11 <= (others => '0'); reg21 <= (others => '0'); reg31 <= (others => '0'); reg41 <= (others =>
 '0');
            reg51 <= (others => '0'); reg61 <= (others => '0'); reg71 <= (others => '0');

            reg12 <= (others => '0'); reg22 <= (others => '0'); reg32 <= (others => '0'); reg42 <= (others =>
 '0');
            reg52 <= (others => '0'); reg62 <= (others => '0'); reg72 <= (others => '0');

            valid8 <= '0'; valid9 <= '0';

        elsif rising_edge(clk) then

            if valid_in='1' then
                valid9 <= valid8;
                valid8 <= reg71(16);

                reg72 <= reg71(15 downto 8);
                reg71(16) <= reg61(16);
                reg71(15 downto 8) <= reg61(7 downto 0);
                reg71(7 downto 0) <= reg62;

                reg62 <= reg61(15 downto 8);
                reg61(16) <= reg51(16);
                reg61(15 downto 8) <= reg51(7 downto 0);
                reg61(7 downto 0) <= reg52;

                reg52 <= reg51(15 downto 8);
                reg51(16) <= reg41(16);
                reg51(15 downto 8) <= reg41(7 downto 0);
                reg51(7 downto 0) <= reg42;

                reg42 <= reg41(15 downto 8);
                reg41(16) <= reg31(16);
                reg41(15 downto 8) <= reg31(7 downto 0);
                reg41(7 downto 0) <= reg32;

                reg32 <= reg31(15 downto 8);
                reg31(16) <= reg21(16);
                reg31(15 downto 8) <= reg21(7 downto 0);
                reg31(7 downto 0) <= reg22;

                reg22 <= reg21(15 downto 8);
                reg21(16) <= reg11(16);
                reg21(15 downto 8) <= reg11(7 downto 0);
                reg21(7 downto 0) <= reg12;

                reg12 <= reg11(15 downto 8);
                reg11(16) <= valid_in;
                reg11(15 downto 8) <= x2;
                reg11(7 downto 0) <= x1;
            else
                valid9 <= '0';
            end if;
        end if;
    end process;

end Behavioral;
```

Code of FIR_parallel_tb Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.math_real.all;

entity FIR_parallel_tb is
end FIR_parallel_tb;

architecture Behavioral of FIR_parallel_tb is

    constant M, data_width, coeff_width : integer := 8;

    component FIR_parallel is
        generic (
                M : integer := 8;
                data_width : integer := 8;
                coeff_width : integer := 8
                );
        Port (
            clk : in std_logic;
            rst : in std_logic;
            valid_in :in std_logic;
            x1 : in std_logic_vector (data_width-1 downto 0);
            x2 : in std_logic_vector (data_width-1 downto 0);
            valid_out : out std_logic;
            fir_out1 : out std_logic_vector(data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0);
            fir_out2 : out std_logic_vector(data_width+coeff_width+integer(ceil(log2(real(M))))-1 downto 0)
            );
    end component;

    signal clk, rst, valid_in, valid_out: std_logic;
    signal x1, x2 : std_logic_vector(data_width-1 downto 0);
    signal fir_out1, fir_out2 : std_logic_vector(data_width+coeff_width+integer(ceil(log2(real(M))))-
1 downto 0);

    constant CLKP : time := 10ns;

    begin
        UUT: FIR_parallel
            port map(
                    clk => clk,
                    rst => rst,
                    valid_in => valid_in,
                    x1 => x1,
                    x2 => x2,
                    valid_out => valid_out,
                    fir_out1 => fir_out1,
                    fir_out2 => fir_out2
                    );

        clk_proc:process
            begin
                clk <= '0';
                wait for CLKP/2;
                clk <= '1';
                wait for CLKP/2;
        end process clk_proc;

        test:process
            begin
                rst <= '1';
                valid_in <= '1';
                x1 <= "00011011";
                x2 <= "11110110";
                wait for CLKP/4;
```

```
                rst<='0';
                wait for 3*CLKP/4;
                valid_in <= '1';
                x1 <= "00000001";
                x2 <= "11000110";
                wait for CLKP;
                x1 <= "11010001";
                x2 <= "11011110";
                wait for CLKP;
                x1 <= "00010101";
                x2 <= "01100110";
                wait for CLKP;
                x1 <= "01000010";
                x2 <= "11001100";
                wait for CLKP;

                x1 <= (others => '0');
                x2 <= (others => '0');

                wait for 88*CLKP;
        end process;
end Behavioral;
```

Οι τιμές για το testbench είναι εκείνες που δόθηκαν κατά την εξέταση και ακολουθεί η επίδειξη της ορθής λειτουργίας:



RESOURCE UTILIZATION:

Resource	Utilization	Available	Utilization %
LUT	209	17600	1.19
LUTRAM	1	6000	0.02
FF	456	35200	1.30
IO	58	100	58.00

TIMING SUMMARY:

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	2	1	CELL17/sum_reg_reg[0]/C	fir_out1[0]	4.076	3.276	0.800
Path 2	∞	2	2	1	CELL17/sum_reg_reg[10]/C	fir_out1[10]	4.076	3.276	0.800
Path 3	∞	2	2	1	CELL17/sum_reg_reg[11]/C	fir_out1[11]	4.076	3.276	0.800
Path 4	∞	2	2	1	CELL17/sum_reg_reg[12]/C	fir_out1[12]	4.076	3.276	0.800
Path 5	∞	2	2	1	CELL17/sum_reg_reg[13]/C	fir_out1[13]	4.076	3.276	0.800
Path 6	∞	2	2	1	CELL17/sum_reg_reg[14]/C	fir_out1[14]	4.076	3.276	0.800
Path 7	∞	2	2	1	CELL17/sum_reg_reg[15]/C	fir_out1[15]	4.076	3.276	0.800
Path 8	∞	2	2	1	CELL17/sum_reg_reg[1]/C	fir_out1[1]	4.076	3.276	0.800
Path 9	∞	2	2	1	CELL17/sum_reg_reg[2]/C	fir_out1[2]	4.076	3.276	0.800
Path 10	∞	2	2	1	CELL17/sum_reg_reg[3]/C	fir_out1[3]	4.076	3.276	0.800

Το critical path σε αυτήν την υλοποίηση είναι ίσο με 4.076ns.

Σύγκριση των τριών υλοποιήσεων (Resources, Critical paths)

Στον παρακάτω πίνακα βλέπουμε συγκεντρωτικά τις παραμέτρους σύμφωνα με τις οποίες θα αξιολογήσουμε τις 3 παραπάνω υλοποιήσεις. Αναφέρεται πως όσον αφορά στα resources στον πίνακα φαίνεται η αριθμητική κατάταξη με αύξουσα σειρά χρησιμοποίησης (λιγότερα resources η FIR – Simple υλοποίηση):

Υλοποίηση	Resource Utilization	Critical Path	Throughput
FIR – Simple	**1**	**6.215ns**	**1/8**
FIR – Pipelined	**2**	**4.076ns**	**1**
FIR – Parallel	**3**	**4.076ns**	**2**

Από τα παραπάνω βλέπουμε ότι υπάρχει ένα αναγκαίο trade – off που θα πρέπει κανείς να αναλογιστεί προκειμένου να επιλέξει την υλοποίηση που τον συμφέρει. Για παράδειγμα, κάποιος που θα ήθελε να εξοικονομήσει υλικό χωρίς να τον ενδιαφέρει η χρονική απόδοση θα τον συνέφερε η πρώτη υλοποίηση ενώ κάποιος που ενδιαφέρεται για γρήγορο και μεγάλο ρυθμό εξόδου τότε θα πρέπει να επιλέξει την τελευταία υλοποίηση.