Ψηφιακά Συστήματα VLSI, 3η Εργαστηριακή Άσκηση

Αγιάννης Κωνσταντίνος 03116352

Ηλιακοπούλου Νικολέτα Μαρκέλα 03116111

Καπελώνης Ελευθέριος 03116163

Ζητούμενο 1

- Κύκλος ρολογιού **6.5 ns**. Συχνότητα λειτουργίας **153MHz**.
- Κατανάλωση πόρων **70 LUTs, 66 Flip Flops, 4 LUTRAM**.

Το πλεονέκτημα αυτής της υλοποίησης είναι ότι έχει μικρό κόστος σε πόρους του FPGA.

```vhdl
-- ghdl -c mac.vhdl ram_example.vhd rom_example.vhd -r FIR_tb --fst=/tmp/out.fst &&
        gtkwave --rcvar 'do_initial_zoom_fit yes' /tmp/out.fst
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;-- this is the only standard. std_logic_unsigned is not


entity MAC is
    generic(
        N: positive := 8;
        L: positive := 20
    );
    port (
        b_in: in unsigned (N-1 downto 0);
        c_in: in unsigned (N-1 downto 0);
        a_out: out unsigned (L-1 downto 0);
        mac_init: in std_logic;
        clock: in std_logic;
        reset: in std_logic
    );
end MAC;

architecture MAC_impl of MAC is
    signal a: unsigned (L-1 downto 0);
begin
    process (clock,reset)
        variable tmp1: unsigned(L-1 downto 0);
        variable tmp2: unsigned(L-1 downto 0);
    begin
        if reset = '1' then
            a_out <= (others => '0');
            a <= (others => '0');
        elsif rising_edge(clock) then
            if mac_init = '1' then
                tmp1 := (others => '0');
            else
                tmp1 := a;
            end if;
            tmp2 := tmp1 + b_in*c_in;
            a <= tmp2; -- avoid tristate logic (inout)
            a_out <= tmp2;
        end if;
    end process;
end MAC_impl;

library IEEE;
use ieee.std_logic_1164.all;

entity delay is
generic
    (
        stages : positive := 4
    );
    port (
        input: in std_logic;
        clock: in std_logic;
        reset: in std_logic;
        output: out std_logic
    );
end delay;

architecture D of delay is
    signal ff: std_logic_vector(stages-1 downto 0);
begin
    output <= ff(0);
    process(clock, reset)
    begin
        if reset = '1' then
            ff <= (others => '0');
        elsif rising_edge(clock) then
            ff(stages-1) <= input;
            for i in 0 to stages-2 loop
                ff(i) <= ff(i+1);
            end loop;
        end if;
    end process;
end D;

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;-- this is the only standard
use std.textio.all;

entity FIR is
    generic(
        -- M: positive := 8;
        N: positive := 8;
        L: positive := 20
    );
    port(
        x: in unsigned(N-1 downto 0);
        y: out unsigned(L-1 downto 0);
        clock: in std_logic;
        reset: in std_logic;
        valid_in: in std_logic;
        valid_out: out std_logic
    );
end FIR;

architecture FIR_impl of FIR is
    signal stage,addr,ram_addr,rom_addr: unsigned(2 downto 0); -- ???
    signal b_mac,c_mac,coeff,ram_di,ram_do: unsigned (N-1 downto 0);
    signal mac_init,ram_en,ram_we,valid_out_delay,mac_init_delay: std_logic;
begin

    mac_unit:entity work.MAC port map(b_in => b_mac,c_in => c_mac,a_out=>y,mac_init =>
        mac_init,clock => clock,reset => reset);
    rom_unit:entity work.mlab_rom port map(clk => clock,addr => rom_addr,rom_out =>
        c_mac,en => '1');

    ram_unit:entity work.mlab_ram port map(clk => clock,addr => ram_addr,we => ram_we,en
        => '1',--ram_en;
    di => ram_di,do =>  b_mac);

    kath: entity work.delay generic map(9) port map(input => valid_out_delay,output =>
        valid_out,clock => clock,reset => reset);

    process (clock,reset)
    begin
        if reset = '1' then
            stage <= "111";
            --valid_out <= '0';
            valid_out_delay <= '0';
            mac_init_delay <= '0';
        elsif rising_edge(clock) then
            -- first 2 cycles are ram[0],rom[1]; ram[0],rom[0]
            -- because ram has 1 cycle update latency
            --valid_out <= valid_out_delay;
            mac_init <= mac_init_delay;
            if stage = 7 then
                if valid_in = '1' then
                    ram_addr <= to_unsigned(0,3);
                    rom_addr <= to_unsigned(1,3);
                    ram_di <= x;
                    ram_we <= '1';
                    stage <= "000";
                    mac_init_delay <= '1';
                    valid_out_delay <= '1';
                else
                    -- valid_out_delay <= '0';
                end if;
            elsif stage = 0 then
                ram_we <= '0';
                ram_addr <= to_unsigned(0,3);
                rom_addr <= to_unsigned(0,3);
                stage <= stage + 1;
                mac_init_delay <= '0';
                valid_out_delay <= '0';
            elsif stage >= 1 then
                rom_addr <= stage + 1;
                ram_addr <= stage + 1;
                stage <= stage + 1;
            end if;
        end if;
    end process;
end FIR_impl;

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;-- this is the only standard

entity FIR_tb is
generic(
        -- M: positive := 8;
        N: positive := 8;
        L: positive := 20;
        inputs_num: positive := 21
    );
end FIR_tb;

architecture test_FIR1 of FIR_tb is
```

```vhdl
    signal x: unsigned(N-1 downto 0);
    signal y: unsigned(L-1 downto 0);
    signal clock: std_logic;
    signal reset: std_logic;
    signal valid_in: std_logic;
    signal valid_out: std_logic;
    constant clock_period: time := 10 ns;
    constant clock_num: integer := 256;
    type inputs_array is array(0 to inputs_num-1) of integer;
    signal inputs: inputs_array := (40, 248, 245, 124, 204, 36, 107, 234, 202, 245, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0);
begin
    unit_to_test:entity work.FIR port map (x => x,y => y, clock => clock, reset => reset,
        valid_in => valid_in, valid_out => valid_out);
    process
    begin
        reset <= '1';
        wait for clock_period;
        reset <= '0';

        unit:for i in 0 to inputs_num-1 loop
            valid_in <= '1';
            x <= to_unsigned(inputs(i),8);
            wait for clock_period;
            valid_in <= '0';
            wait for clock_period;
            wait for clock_period;
            wait for clock_period;
            wait for clock_period;
            wait for clock_period;
            wait for clock_period;
            wait for clock_period;
        end loop;

        wait;
    end process;
    clocking: process
    begin
        for i in 0 to clock_num loop
            clock <= '1', '0' after clock_period / 2;
            wait for clock_period;
        end loop;
    wait;
    end process;
end test_FIR1;
```

Υλοποίηση της RAM

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.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
          addr : in unsigned(2 downto 0);           -- memory address
          di   : in unsigned(data_width-1 downto 0);        -- input data
          do   : out unsigned(data_width-1 downto 0));      -- output data
end mlab_ram;
```

```vhdl
architecture Behavioral of mlab_ram is
    type ram_type is array (7 downto 0) of unsigned (data_width-1 downto 0);
    signal RAM : ram_type := (others => (others => '0'));
begin

    process (clk)
    begin
        if rising_edge(clk) then
            if en = '1' then
                if we = '1' then                -- write operation
                    RAM(0) <= di;
                    for i in 0 to 6 loop
                        RAM(i+1) <= RAM(i);
                    end loop;
                end if;
                do <= RAM(to_integer(addr));
            end if;
        end if;
    end process;

end Behavioral;
```

Ζητούμενο 2

Στο ζητούμενο αυτό ξεδιπλώνουμε το φίλτρο και το μετατρέπουμε σε πλήρως pipelined. Μπορούμε να πούμε ότι "σπάμε" τη διαδικασία σε 8 μεγάλα στάδια καθένα από τα οποία αποτελείται από την είσοδο,την δημιουργία του αντίστοιχου γινομένου και την άθροιση αυτού με το προηγούμενο άθροισμα(με το 0 αν είμαστε στο πρώτο στάδιο). Προσθέτουμε καταχωρητές σε κάθε στάδιο εισόδου, μετά από τη δημιουργία ενός γινομένου και μετά από τη δημιουργία ενός αθροίσματος. Για τον κατάλληλο συγχρονισμό προστίθεται σε κάθενα από τα 8 στάδια που περιγράψαμε, ένας ακόμα καταχωρητής στην είσοδο που λειτουργεί ως delay. Δεδομένου ότι όλες οι είσοδοι έχουν αρχικοποιηθεί με 0, παράγεται αποτέλεσμα μετά από latency 10 κύκλων.

Το πλεονέκτημα της συγκεκριμένης υλοποίησης σε σχέση με το ζητούμενο 1 είναι ότι παίρνουμε αποτέλεσμα σε κάθε κύκλο ρολογιού. Το κρίσιμο μονοπάτι μειώνεται, κι έτσι αυξάνεται η συχνότητα λειτουργίας του fpga, γεγονός που αυξάνει και την απόδοση. Το μειωνέκτημα, ωστόσο, της συγκεκριμένης υλοποίησης είναι ότι χρειάζεται αρκετά μεγαλύτερος αριθμός πόρων FPGA, 8πλασιος περίπου (8 αθροιστές, 8 πολλαπλασιαστές και 4 καταχωρητές σε κάθε στάδιο)

Μετά τη σύνθεση του κυκλώματος βρίσκουμε το unconstrained critical path. * Max Total Delay **4.076 ns**. Άρα * Κύκλος ρολογιού **4.076 ns**. Συχνότητα λειτουργίας **245.33MHz**. * Κατανάλωση πόρων **120 LUTs, 289 Flip Flops, 9 LUTRAM**.

Κώδικας του pipelined_fir

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE ieee.math_real."ceil";
USE ieee.math_real."log2";

PACKAGE types IS

    CONSTANT taps        : INTEGER := 8; --number of fir filter taps
    CONSTANT data_width  : INTEGER := 8; --width of data input
    CONSTANT coeff_width : INTEGER := 8; --width of coefficients
    CONSTANT ZERO : UNSIGNED(data_width + coeff_width + integer(ceil(log2(real(taps)))) -
        1  downto 0) := (others => '0');
    TYPE coefficient_array IS ARRAY (0 TO taps-1) OF STD_LOGIC_VECTOR(coeff_width-1
        DOWNTO 0);  --array of all coefficients
    CONSTANT COEFS : coefficient_array := ("00000001", "00000010", "00000011",
        "00000100", "00000101", "00000110", "00000111", "00001000");
    TYPE data_array IS ARRAY (0 TO 2*taps-1) OF UNSIGNED(data_width-1 DOWNTO 0);
        --array of historic data values, latency*2 for sync
    TYPE product_array IS ARRAY (0 TO taps-1) OF UNSIGNED((data_width + coeff_width)-1
        DOWNTO 0); --array of coefficient * data products
    TYPE ADD_TYPE IS ARRAY(0 TO taps-1) of UNSIGNED(data_width + coeff_width +
        integer(ceil(log2(real(taps)))) - 1 downto 0); --array of sums of products*coef
    TYPE valid_array IS ARRAY(0 TO 17) of UNSIGNED(0 DOWNTO 0); --length same as latency-
        1 (1 stage before result)

END PACKAGE types;

LIBRARY ieee;
LIBRARY xil_defaultlib;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;
USE ieee.math_real."ceil";
USE ieee.math_real."log2";
USE work.types.all;

ENTITY fir_filter IS
    PORT(
            valid_in        :   IN  STD_LOGIC_VECTOR(0 DOWNTO 0);
            clk             :   IN  STD_LOGIC;                                  --system
        clock
            reset_n         :   IN  STD_LOGIC;                                  --active
        low asynchronous reset
            data            :   IN  STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);    --data
        stream
--          data_pipeline   :   INOUT data_array;
            result          :   OUT STD_LOGIC_VECTOR((data_width + coeff_width +
        integer(ceil(log2(real(taps)))) - 1) DOWNTO 0);
--          ADD             :   INOUT  ADD_TYPE;  --filtered result
            valid_out       :   OUT STD_LOGIC_VECTOR(0 DOWNTO 0));
--          valid_reg       :   INOUT valid_array;
--          count           :   INOUT STD_LOGIC_VECTOR(2 downto 0):="000");

END fir_filter;

ARCHITECTURE behavior OF fir_filter IS
    SIGNAL products         : product_array:=(OTHERS => (OTHERS => '0'));     --array of
        coefficient*data products
      SIGNAL valid_reg        : valid_array:=(OTHERS => (OTHERS => '0'));
      SIGNAL data_pipeline    : data_array:=(OTHERS => (OTHERS => '0'));        --
        pipeline of historic data values
      SIGNAL ADD                :  ADD_TYPE:=(OTHERS => (OTHERS => '0'));
    ATTRIBUTE syn_multstyle : string;
    ATTRIBUTE syn_multstyle OF products : SIGNAL IS "logic";
    SIGNAL count : STD_LOGIC_VECTOR(2 downto 0):="000";

BEGIN

    PROCESS(clk, reset_n)

    BEGIN

        IF(reset_n = '1') THEN                                      --asynchronous reset

            data_pipeline <= (OTHERS => (OTHERS => '0'));               --clear data
        pipeline values
            valid_reg <= (OTHERS => (OTHERS => '0'));
            products <= (OTHERS => (OTHERS => '0'));
            ADD <= (OTHERS => (OTHERS => '0'));
            result <= (OTHERS => '0');                              --clear result
        output
            valid_out <= (OTHERS => '0');
        ELSIF(clk'EVENT AND clk = '1') THEN                       --not reset

            data_pipeline <= UNSIGNED(data) & data_pipeline(0 TO 2*taps-2); --shift new
        data into data pipeline
                                                                        --even index
        is extra latency, odd index shifts to product array
            IF valid_in = "1" THEN
             IF  count > "000" THEN --there is previous valid_in=0 that affects validity
        of final result
                 valid_reg <= UNSIGNED(not(valid_in)) & valid_reg(0 TO 16); --shift
        inversed new valid_in (0) into valid_reg
                 count <= count-1 ; --previous valid_in=0 goes to next stage
             ELSIF count="000" THEN --previous valid_in = 0 does not exist anymore
                 valid_reg<= UNSIGNED(valid_in) & valid_reg(0 TO 16);
             END IF;
            ELSIF valid_in = "0" THEN
              valid_reg <= UNSIGNED(valid_in) & valid_reg(0 TO 16); --shift new valid_in
        into valid_reg
              count <= "111" ; --update counter, new valid_in=0 that affects "this" and
        the next 7 final results' validity
            END IF;
            FOR i IN taps-1 DOWNTO 0 LOOP
             IF i = 0 THEN
                 ADD(i) <=  products(i) + ZERO ;
             ELSE
                 ADD(i) <=  products(i) + ADD(i-1);
             END IF;
             products(i) <= data_pipeline(2*i + 1) * UNSIGNED(COEFS(i));
            END LOOP;
            result <= STD_LOGIC_VECTOR(ADD(taps-1));                         --
        output result
            valid_out <= STD_LOGIC_VECTOR(valid_reg(17));                    --
        valid_out

        END IF;
    END PROCESS;


END behavior;
```

Κώδικας του testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.math_real."ceil";
use IEEE.math_real."log2";
use work.types.all;

entity fir_filter_tb is
end fir_filter_tb;

architecture Behavioral of fir_filter_tb is

        signal  valid_in     : std_logic_vector(0 DOWNTO 0);
        signal  clk          : std_logic:='1';
        signal  rst          : std_logic;
        signal  data         : std_logic_vector(data_width - 1  DOWNTO 0);
--      signal  data_pipeline: data_array;
```

```vhdl
        signal  result       : STD_LOGIC_VECTOR((data_width + coeff_width +
        integer(ceil(log2(real(taps)))) - 1) DOWNTO 0);
--        signal  ADD          : ADD_TYPE;
        signal  valid_out    : std_logic_vector(0 DOWNTO 0);
--        signal  valid_reg    : valid_array;
--        signal  count        : std_logic_vector(2 downto 0):="000";

        constant clock_period: time := 10 ns;
        constant clock_num: integer := 2048;
begin
    UUT: entity work.fir_filter port map (valid_in=>valid_in, clk=>clk, reset_n=>rst,
        data=>data,
     result=>result, valid_out=>valid_out);

-- Process for generating the clock
    clk <= not clk after clock_period / 2;

process is
 begin
 rst <= '1';
 valid_in <= "0";
 wait for clock_period;
    rst <= '0';
    valid_in <= "0";

    data <= std_logic_vector(to_unsigned(40,8));
            wait for clock_period;
    data <= std_logic_vector(to_unsigned(248,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(245,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(124,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(204,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(36,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(107,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(234,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(202,8));
            wait for clock_period;
    data <= std_logic_vector(to_unsigned(245,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(0,8));
            wait for clock_period;


    data <= std_logic_vector(to_unsigned(5,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(12,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(33,8));
            wait for clock_period;
            valid_in <= "1";
            data <= std_logic_vector(to_unsigned(28,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(6,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(2,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(17,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(9,8));
            wait for clock_period;
            data <= std_logic_vector(to_unsigned(1,8));
            wait for clock_period;
    for i0 in 0 to 1 loop
        for i3 in 0 to 5 loop
            data <= std_logic_vector(to_unsigned((i3 mod 8),8));
            wait for clock_period;
            end loop;
        valid_in <= "0";
        data <= std_logic_vector(to_unsigned(6,8));
        wait for clock_period;
        valid_in <= "1";
        data <= std_logic_vector(to_unsigned(7,8));
        wait for clock_period;
        for i4 in 8 to 20 loop
            data <= std_logic_vector(to_unsigned((i4 mod 8),8));
            wait for clock_period;
            end loop;
        -- rst <= not rst;
        end loop;
    wait;
end process;



end Behavioral;
```

Ζητούμενο 3

Η συγκεκριμένη αρχιτεκτονική είναι σχεδόν ίδια με αυτή του ζητούμενου 2. Τώρα όμως υπάρχουν 2 παράλληλες σειρές πολλαπλασιαστών-αθροιστών. Η κάθε μία παράγει μία από τις εξόδους που αντιστοιχούν στην τρέχουσα χρονική στιγμή μετά από ένα latency 10 κύκλων. Για να υπάρχει ο σωστός συγχρονισμός προστίθεται ένας αριθμός από registers (που λειτουργούν ως delay) στην έξοδο των πολλαπλασιαστών. Συγκεκριμένα ο πρώτος πολλαπλασιαστής έχει 0 delays, ο δεύτερος 1 delay κλπ.

Το πλεονέκτημα της συγκεκριμένης υλοποίησης είναι ότι έχουμε διπλάσιο throughput από την προηγούμενη αφού παράγονται δύο αποτελέσματα ανά κύκλο ρολογιού. Ωστόσο το κόστος είναι ο μεγαλύτερος αριθμός πόρων FPGA που χρειάζονται.

- Κύκλος ρολογιού **2.626 ns**. Συχνότητα λειτουργίας **380MHz**.
- Κατανάλωση πόρων **395 LUTs, 729 Flip Flops, 79 LUTRAM**.

Κώδικας του parallel_fir

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.math_real."ceil";
use IEEE.math_real."log2";

package parallel_fir_types is
    constant P : positive := 2;
    constant N : positive := 8;
    constant M : positive := 8;
    constant logM : integer := integer(ceil(log2(real(M))));
    constant Y_BITS : positive := M + N + logM; --19
    constant ZERO : unsigned(Y_BITS - 1  downto 0) := (others => '0');

    type x_type is array(P-1 downto 0) of unsigned(N-1 downto 0);
    type y_type is array(P-1 downto 0) of unsigned(Y_BITS-1 downto 0);
    type x_reg_type is array(M+P-1 downto 0) of unsigned(N-1 downto 0);
    type add_reg_type is array(M-1 downto 0) of unsigned(Y_BITS-1 downto 0);
    type twod_add_reg_type is array(P-1 downto 0) of add_reg_type;
    type product_reg_type is array(M-1 downto 0) of unsigned(N+N-1 downto 0);
    type twod_product_reg_type is array(P-1 downto 0) of product_reg_type;
    type coeff_type is array(M-1 downto 0) of unsigned(N-1 downto 0);

    constant COEFF : coeff_type := ("00001000", "00000111", "00000110", "00000101",
        "00000100", "00000011", "00000010", "00000001");
end package;

library IEEE;
use ieee.std_logic_1164.all;

entity delay is
generic
    (
        stages : positive := 4
    );
    port (
        input: in std_logic;
        clock, valid_in: in std_logic;
        rst: in std_logic;
        output: out std_logic
    );
end delay;

architecture D of delay is
signal ff: std_logic_vector(stages-1 downto 0);
begin
    output <= ff(0);
    process(clock, valid_in, rst)
    begin
        if rst = '1' then
            ff <= (others => '0');
        elsif rising_edge(clock) and valid_in = '1' then
            ff(stages-1) <= input;
            for i in 0 to stages-2 loop
                ff(i) <= ff(i+1);
            end loop;
        end if;
    end process;
end D;

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

use work.parallel_fir_types.all;

entity parallel_fir is
    port (
        x : in x_type;
        valid_in : in std_logic;
        rst : in std_logic;
        clk : in std_logic;
        valid_out : out std_logic;
        y : out y_type
    );
end parallel_fir;

architecture Behavioral of parallel_fir is
    signal x_reg: x_reg_type := (others => (others => '0'));
    signal y_reg : y_type;
    signal valid_out_cnt : unsigned(3 downto 0) := (others => '0');
    signal add_reg : twod_add_reg_type := (others => (others => (others => '0')));
    signal product_reg, product_delayed : twod_product_reg_type := (others => (others =>
        (others => '0')));
begin

-- add delays (flip-flops) after the multipliers in order to synchronize the pipeline
l1: for K in 0 to P-1 generate
l2: for I in 0 to M-1 generate
l3: for J in 0 to N+N-1 generate
    l4: if I>0 generate
        p1: entity work.delay
        generic map(I) port map(input => product_reg(K)(I)(J), output =>
        product_delayed(K)(I)(J), clock => clk, valid_in => valid_in, rst => rst);
    end generate;
    l5: if I=0 generate
        product_delayed(K)(I)(J) <= product_reg(K)(I)(J);
    end generate;
end generate;
end generate;
end generate;

process (clk, rst, valid_in)
begin
    if rst = '1' then
        x_reg <= (others => (others => '0'));
        y_reg <= (others => (others => '0'));
        add_reg <= (others => (others => (others => '0')));
        y <= (others => (others => '0'));
        valid_out_cnt <= to_unsigned(0, valid_out_cnt'length);
        valid_out <= '0';
    elsif clk'event and clk = '1' then
        if valid_in = '1' then
            -- intermediate registers
            -- P parallel pipelines
            for K in P-1 downto 0 loop
                for I in M-1 downto 0 loop
                    if I = 0 then
                        add_reg(K)(I) <= zero + resize(product_delayed(K)(I), Y_BITS);
                    else
                        add_reg(K)(I) <= resize(product_delayed(K)(I) + add_reg(K)(I-1),
        Y_BITS);
                    end if;
                    product_reg(K)(I) <= x_reg(I+K) * COEFF(I);
                end loop;
            end loop;
           for J in M-1+P-1 downto 0 loop
                    if J < P then
                        x_reg(J) <= x(J);
                    else
                        x_reg(J) <= x_reg(J-P);
                    end if;
            end loop;

            for K in 0 to P-1 loop
                y(K) <= add_reg(K)(M-1);
            end loop;

            -- valid out
            if valid_out_cnt = 10 then -- latency : 10
                valid_out <= '1';
            else
                valid_out_cnt <= valid_out_cnt + 1;
                valid_out <= '0';
            end if;
        else
            valid_out <= '0';
        end if;
    end if;
end process;

end Behavioral;
```

Κώδικας του testbench

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

entity parallel_fir_tb is
end parallel_fir_tb;

use work.parallel_fir_types.all;

architecture Behavioral of parallel_fir_tb is
-- inputs
    signal x : x_type;
    signal valid_in : std_logic;
    signal rst : std_logic;
    signal clk : std_logic;
-- outputs
    signal y : y_type;
    signal valid_out : std_logic;

    constant clock_period: time := 10 ns;
    constant clock_num: integer := 2048;
begin
    UUT: entity work.parallel_fir port map (x=>x, valid_in=>valid_in, rst=>rst, clk=>clk,
        y=>y, valid_out=>valid_out);

process begin
    -- trivial tests
    rst <= '0';
    valid_in <= '1';
    for rst_val in std_logic range '0' to '1' loop
        for valid_in_val in std_logic range '0' to '1' loop
            rst <= rst_val;
            valid_in <= valid_in_val;
            x(1) <= "00000001";
```

```
                x(0) <= "00000000";
                wait for clock_period;
                x(1) <= "00000000";
                x(0) <= "00000000";
                wait for clock_period;
                x(1) <= "00000000";
                x(0) <= "00000000";
                wait for clock_period;
                x(1) <= "00000000";
                x(0) <= "00000000";
                wait for clock_period;
                x(1) <= "00000000";
                x(0) <= "00000000";
                wait for 10*clock_period;
            end loop;
        end loop;

        rst <= '1';
        wait for 1 ns;
        rst <= '0';
        valid_in <= '1';
        x(1) <= to_unsigned(40, 8);
        x(0) <= to_unsigned(248, 8);
        wait for clock_period;
        x(1) <= to_unsigned(245, 8);
        x(0) <= to_unsigned(124, 8);
        wait for clock_period;
        x(1) <= to_unsigned(204, 8);
        x(0) <= to_unsigned(36, 8);
        wait for clock_period;
        x(1) <= to_unsigned(107, 8);
        x(0) <= to_unsigned(234, 8);
        wait for clock_period;
        x(1) <= to_unsigned(202, 8);
        x(0) <= to_unsigned(245, 8);
        wait for clock_period;
        x(1) <= to_unsigned(0, 8);
        x(0) <= to_unsigned(0, 8);
        wait for clock_period;
        x(1) <= to_unsigned(0, 8);
        x(0) <= to_unsigned(0, 8);
        wait for clock_period;
        x(1) <= to_unsigned(0, 8);
        x(0) <= to_unsigned(0, 8);
        wait for clock_period;
        x(1) <= to_unsigned(0, 8);
        x(0) <= to_unsigned(0, 8);
        wait for 20*clock_period;
        wait;
    end process;

    clocking: process
        begin
            for i in 0 to clock_num loop
                clk <= '1', '0' after clock_period / 2;
                wait for clock_period;
            end loop;
        wait;
        end process;
    end Behavioral;
```