5η εργαστηριακή άσκηση

Φωτεινή Κυριακοπούλου 03120182
Ιωάννης Μπουφίδης 03120162

FIR Φίλτρο στο ZYBO

Για την υλοποιήση του IP του FIR φίλτρου, χρησιμοποιήσαμε το φίλτρο της προηγούμενης άσκησης με κάποιες αλλαγές, προκειμένου να ανταποκρίνεται στις συνθήκες της άσκησης. Συγκεκριμένα, τροποιήσαμε τις εξής δομικές μονάδες:

- **MAC - Μονάδα Πολλαπλασιασμού με Συσσώρευση**

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

entity mac_unit is
    port (
        clk : in std_logic;
        rom_out : in std_logic_vector(7 downto 0); -- Coefficients from ROM
        ram_out : in std_logic_vector(7 downto 0); -- Input data from RAM
        mac_init : in std_logic;                   -- Initialization signal
        valid_out_tmp: in std_logic;
        valid_out: out std_logic := '0';
        y_out : inout std_logic_vector(18 downto 0) -- Output
    );
end entity mac_unit;

architecture mac_unit_arch of mac_unit is
 signal sum: std_logic_vector(18 downto 0);
 signal sig_valid: std_logic;
begin
    process (clk)
    begin
        if rising_edge(clk) then
            if mac_init = '1' then
                -- Reset accumulator
                sum <= "000" & rom_out*ram_out;
                valid_out <= '0';
            elsif mac_init = '0' then
                sum <= sum + ram_out*rom_out; -- Add to partial sum computed product
            end if;
            sig_valid <= valid_out_tmp;
            valid_out <= sig_valid;
            if sig_valid='1' then y_out <= sum; end if;
        end if;
    end process;
end mac_unit_arch;
```

5η εργαστηριακή άσκηση

- **Control Unit - Μονάδα Ελέγχου**

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

entity control_unit is
    port (
        clk : in std_logic;
        reset : in std_logic;
        valid_in: in std_logic;
        mac_init : out std_logic;
        valid_out_tmp: out std_logic;
        we, en: out std_logic;
        rom_address : out std_logic_vector(2 downto 0);
        ram_address : out std_logic_vector(2 downto 0)
    );
end entity control_unit;

architecture control_unit_arch of control_unit is
    signal counter : std_logic_vector(2 downto 0):="000";  -- Counter for ROM and RAM
begin
    process (clk, reset)
    begin
        if reset = '1' then
            counter <= "000"; -- Reset counter
            valid_out_tmp <= '0'; -- Invalid output
        elsif rising_edge(clk) then
            if counter="000" then
                valid_out_tmp <= '0';
                if valid_in='1' then
                    mac_init <= '1'; -- Initialize mac unit
                    we <= '1'; en <= '1'; -- Write x to RAM[0]
                    rom_address <= std_logic_vector(counter);
                    ram_address <= std_logic_vector(counter);
                    counter <= counter + 1;
                else
                    we <= '0'; en <= '0'; -- Input is invalid, disable access to RAM
                    mac_init <= '1';
                end if;
             else
                we <= '0'; en <= '1';  -- Read RAM[1-7]
                mac_init <= '0';
                rom_address <= std_logic_vector(counter);
                ram_address <= std_logic_vector(counter);
                if counter="111" then valid_out_tmp <= '1'; -- Valid output
                else valid_out_tmp <= '0'; end if;
                counter <= counter + 1;
            end if;
        end if;
    end process;
end architecture control_unit_arch;
```

5η εργαστηριακή άσκηση

- **FIR Φίλτρο**

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

entity fir_filter is
  port (
        clk, reset, valid_in: in std_logic;
        x: in std_logic_vector(7 downto 0);      -- Filter input
        valid_out: out std_logic;
        y: inout std_logic_vector(18 downto 0)  -- Filter output
        );
end fir_filter;

architecture fir_filter_arch of fir_filter is
    component mac_unit is
    port (
        clk : in std_logic;
        rom_out : in std_logic_vector(7 downto 0);
        ram_out : in std_logic_vector(7 downto 0);
        mac_init : in std_logic;
        valid_out: out std_logic;
        valid_out_tmp: in std_logic;
        y_out : inout std_logic_vector(18 downto 0)
    );
    end component;

    component control_unit is
    port (
        clk : in std_logic;
        reset : in std_logic;
        valid_in: in std_logic;
        mac_init : out std_logic;
        we, en: out std_logic;
        valid_out_tmp: out std_logic;
        rom_address : out std_logic_vector(2 downto 0);
        ram_address : out std_logic_vector(2 downto 0)
    );
    end component;

    component mlab_ram is
    generic (
        data_width : integer :=8
    );
    port (clk  : in std_logic;
          reset: in std_logic;
          we   : in std_logic;
          en   : in std_logic;
          addr : in std_logic_vector(2 downto 0);
          di   : in std_logic_vector(data_width-1 downto 0);
```

```
            do   : out std_logic_vector(data_width-1 downto 0));
    end component;

    component mlab_rom is
    generic (
        coeff_width : integer :=8
     );
    port ( clk : in  STD_LOGIC;
           en : in  STD_LOGIC;
           addr : in  STD_LOGIC_VECTOR (2 downto 0);
           rom_out : out  STD_LOGIC_VECTOR (coeff_width-1 downto 0));
    end component;

    component D1 is
    port (
          reset, Clk, D: in std_logic;
          Q: out std_logic
         );
    end component;

    signal rom_out, ram_out: std_logic_vector(7 downto 0);
    signal rom_ad, ram_ad: std_logic_vector(2 downto 0);
    signal mac_i, mac_ii, we, en, pr_i, pr_ii: std_logic;
begin
    control: control_unit port map(clk => clk, reset => reset, valid_in => valid_in,
                                   mac_init => mac_i, rom_address => rom_ad,
                                   ram_address => ram_ad, we => we, en => en,
                                   valid_out_tmp => pr_i);
    valid_ff: D1 port map (clk => clk, D => pr_i, Q => pr_ii, reset => reset);
    mac_in_delay: D1 port map (clk => clk, D => mac_i, Q => mac_ii, reset => '0');
    -- Delay mac_init, valid_out_tmp signal one cycle to synchronize operations
    ram: mlab_ram generic map(data_width => 8)
         port map(clk => clk, reset => reset,
                  we => we, en => en, addr => ram_ad, di => x, do => ram_out);
    rom: mlab_rom generic map(coeff_width => 8)
         port map(clk => clk, en => en, addr => rom_ad, rom_out => rom_out);
    mac: mac_unit port map (clk => clk, rom_out => rom_out, ram_out => ram_out,
                            valid_out_tmp => pr_ii, valid_out => valid_out,
                            mac_init => mac_ii, y_out => y);
end fir_filter_arch;
```

Συγκεκριμένα, για το καλύτερο συγχρονισμό της εξόδου του συστήματος, το σήμα valid_out ενεργοποιείται κάθε φορά στο Control Unit κατά την διάρκεια της τελευταίας επανάληψης μίας ισχύουσας (valid_in = ‘1’) πράξης.

Στην συνέχεια, μετά από καθυστέρηση 2 κύκλων (1 στο D Flip-Flop, 1 στη MAC), το σήμα εξόδου valid_out ενεργοποιείται, ανανεώνοντας ταυτόχρονα την έξοδο y του συστήματος με το τελευταίο έγκυρο αποτέλεσμα που παράχθηκε.

5η εργαστηριακή άσκηση

Προκειμένου να δημιουργηθεί το IP του FIR ως AXI4-Lite Slave, χρειάστηκε να προστεθούν (ή να επεξεργαστούν) τα παρακάτω σημεία του ήδη υπάρχοντος AXI.

- **Ορισμός απαραίτητων σημάτων και component fir_filter**

```
signal input : std_logic_vector(31 downto 0);  --input=0..0&rst&valid_in&x
signal output : std_logic_vector(31 downto 0); --output= 0..0&valid_out_ip&y_ip

signal valid_out_ip: std_logic;
signal y_ip: std_logic_vector(18 downto 0);

 component fir_filter is
        Port ( clk : in std_logic;
               reset : in std_logic;
               valid_in : in std_logic;
               x : in std_logic_vector(7 downto 0);
               valid_out : out std_logic;
               y : out std_logic_vector (18 downto 0)
               );
 end component;
```

- **Συγχρονισμός αποστολής δεδομένων προς τον ARM**

```
    -- Output register or memory read data
    process( S_AXI_ACLK ) is
    begin
      if (rising_edge (S_AXI_ACLK)) then
        if ( S_AXI_ARESETN = '0' ) then
          axi_rdata  <= (others => '0');
        else
            -- When there is a valid read address (S_AXI_ARVALID) with
            -- acceptance of read address by the slave (axi_arready),
            -- output the read dada
            -- Read address mux
                if( valid_out_ip = '1')   then
                    slv_reg1 <= output;       --send result when valid_out = '1'
                elsif(slv_reg_rden = '1') then
                   axi_rdata <= reg_data_out; --slv_reg1;
                   slv_reg1(19) <= '0';       --else set valid_out = '0'
                else  slv_reg1 <= slv_reg1;
            end if;
          end if;
        end if;
    end process;
```

- **Αντιστοίχηση σημάτων-ports**

```
    -- Add user logic here
          FIR: fir_filter  port map (
                 clk => S_AXI_ACLK,
                 reset => input(9),
                 x => input(7 downto 0),
                 valid_in => input(8),
                 valid_out => valid_out_ip,
                 y => y_ip
                 );

          process (S_AXI_ACLK) is
                begin
                      input <= slv_reg0;  -- read input from slv_reg0
                      output <= "000000000000"&valid_out_ip & y_ip;
                end process;
    -- User logic ends
```

Προκειμένου να ελεγχθεί η σωστή λειτουργία του συστήματος στο ZYBO, υλοποιήθηκε το εξής C πρόγραμμα:

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "sleep.h"
#include <inttypes.h>
#include "xil_types.h"
#include "xil_io.h"
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "sleep.h"
#define MY_IP_BASEADDR 0x43C00000

int main() {
    init_platform();

      unsigned int input, output;
      unsigned int data_in, valid_in, reset;
      unsigned int valid_out, result;
      _Bool rst;

     while(1) {

       valid_out =0;
```

```
        xil_printf("Give input data(0-255):\n");
        scanf("%d",&data_in);

        xil_printf("Do you want to reset?\n");
        scanf("%d",&reset);

        xil_printf("Is input valid?\n");
        scanf("%d",&valid_in);

        valid_in = valid_in << 8;
        rst = reset;
        reset = reset << 9;
        input = reset | valid_in | data_in;

        xil_printf("Input data is:%d\n", data_in);

        Xil_Out32((MY_IP_BASEADDR+0x00), input);

           if ( rst || valid_in == 0)  {
               if (valid_in == 0) xil_printf("Invalid input\n");
               else xil_printf("Reseted!\n");
                  }
          else  {
            while (valid_out == 0 ) {
                   output = Xil_In32(MY_IP_BASEADDR+0x04);
                   valid_out = output &  0x80000;
                }


                result = output & 0x7FFFF;
                xil_printf("Result is: %d\n", result);
            }

     }
    cleanup_platform();
    return 0;
}
```