

COL215 Assignment 3

Title: Digital Image Filter

Introduction:

This assignment is an exercise in designing a circuit for performing simple computation over data stored in memory. It involves integration of pre-designed memories and multiplier-accumulator components in your design.

Specification:

Design an image filter in VHDL that uses a 3x3 sliding window for QQVGA size images (look up Wikipedia page on “Graphic Display Resolutions” for meaning of QQVGA), with 8-bit pixel resolution.

Use a Read-Write Memory module (RWM, more popularly known as RAM or Random-Access Memory) to store the original image as well as the filtered image and a Read Only Memory module (ROM) to store the filter coefficients. The coefficient memory should hold two sets of coefficients - one for a *smoothing filter* and one for a *sharpening filter*. Do not bother about how the memory modules are initialized with original image and filter coefficients.

An external switch indicates whether the image is to be smoothed or sharpened. A push-button is used to start the filtering operation. After constructing the filtered image, the system waits for next pressing of the push-button.

Entity and Architecture declarations for the memory modules and a multiplier-accumulator are given in the Annexure. You need to create a single instance of each of these and use in your design to meet all your memory and multiplication requirements.

You need to submit: Design overview (text), ASM chart, VHDL code with explanation.

Details:

A. Filtering Operation

Let X be the $m \times n$ matrix denoting the image to be filtered.

Let Y be the $m-2 \times n-2$ matrix denoting the filtered image.

Let C be the 3×3 filter coefficient matrix.

The filtering operation is defined as follows.

$$Y[I, J] = \sum_{i=-1}^1 \sum_{j=-1}^1 X[I + i, J + j] * C[i, j] \\ 1 \leq I \leq m-2, 1 \leq J \leq n-2$$

The elements of X are read from the image memory one by one, multiplied by coefficients read from coefficient memory and summed. The sum forms one element of Y and is stored in memory.

B. Filtering examples

Two examples of coefficient matrices to be considered are shown here and effect of filtering with these coefficients is illustrated. Sum of the coefficients is usually kept as 1.

Original 120 x 160 image
Pixel values: 0 to 255



Smoothing filter :

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$



Sharpening filter :

$$\begin{bmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{1}{3} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{bmatrix}$$



C. Handling fractions

The coefficients can be scaled up by a suitable power of 2 and rounded off to integers. The final results could be scaled down by the same factor, simply by doing a right shift. The pixel values in X are 8 bit unsigned numbers in the range 0 to 255. These can be treated as 9 bit signed integers by prefixing a 0. Keeping the above two filter examples in mind, let

the scaling factor be $2^7 = 128$. Then coefficient values can also be treated as 9 bit signed integers. Use 18 bits for holding the products and carrying out summation. In the final result, replace negative values with 0 and perform down scaling, that is, right shift by 7 bits (in other words, discard the rightmost 7 bits). In the above examples, the result will not exceed 9 bits (in other words, the leftmost 2 bits can be discarded). Note that scaling up of the coefficients is done off-line (these are constants that can be pre-computed) whereas scaling down of the result is done in the circuit.

Annexure: Pre-designed Modules

Three pre-designed modules are described here -

- A read-write memory called RAM_64Kx8, having 64K words, 8 bit wide. This is to be used for storing the original image at address 0 (0000_{16}) onwards and the filtered image at address 32768 (8000_{16}).
- A read-only memory called ROM_32x9, having 32 words, 9 bit wide. This to be used for storing coefficients of one filter at address 0 onwards and those of the other filter at address 16 onwards
- A multiplier-accumulator called MAC, containing a 18x18 multiplier, an 18-bit register and an 18-bit adder to perform sum-of-product (arithmetic, not logical) computation required for filtering.

The entity-architecture description of these modules is modeled after the actual building blocks available in Artix 7 series FPGAs. BASYS3 board used in the lab has an Artix 7 series FPGA.

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

entity RAM_64Kx8 is

```
  port (
    clock : in std_logic;
    read_enable, write_enable : in std_logic;  -- signals that enable read/write operation
    address : in std_logic_vector(15 downto 0);  --  $2^{16} = 64K$ 
    data_in : in std_logic_vector(7 downto 0);
    data_out : out std_logic_vector(7 downto 0)
  );
```

end RAM_64Kx8;

entity ROM_32x9 is

```
  port (
    clock : in std_logic;
    read_enable : in std_logic;  -- signal that enables read operation
    address : in std_logic_vector(4 downto 0);  --  $2^5 = 32$ 
    data_out : out std_logic_vector(7 downto 0)
  );
```

end ROM_32x9;

entity MAC is

```
  port (
    clock : in std_logic;
    control : in std_logic;  -- '0' for initializing the sum
    data_in1, data_in2 : in std_logic_vector(17 downto 0);
    data_out : out std_logic_vector(17 downto 0)
  );
```

end MAC;

architecture Artix of RAM_64Kx8 is

type Memory_type is array (0 to 65535) of std_logic_vector (7 downto 0);

signal Memory_array : Memory_type;

begin

process (clock) begin

if rising_edge (clock) then

if (read_enable = '1') then -- the data read is available after the clock edge

data_out <= Memory_array (to_integer (unsigned (address)));

end if;

if (write_enable = '1') then -- the data is written on the clock edge

Memory_array (to_integer (unsigned (address))) <= data_in;

end if;

end if;

end process;

end Artix;

architecture Artix of ROM_32x9 is

type Memory_type is array (0 to 31) of std_logic_vector (8 downto 0);

signal Memory_array : Memory_type;

begin

process (clock) begin

if rising_edge (clock) then

if (read_enable = '1') then -- the data read is available after the clock edge

data_out <= Memory_array (to_integer (unsigned (address)));

end if;

end if;

end process;

end Artix;

architecture Artix of MAC is

signal sum, product : signed (17 downto 0);

begin

data_out <= std_logic_vector (sum);

product <= signed (data_in1) * signed (data_in2)

process (clock) begin

if rising_edge (clock) then

 -- sum is available after clock edge

if (control = '0') then

 -- initialize the sum with the first product

sum <= std_logic_vector (product);

else

 -- add product to the previous sum

sum <= std_logic_vector (product + signed (sum));

end if;

end if;

end process;

end Artix;

Note on COL215 Assignment 3

This note is in response to some queries that I have received from students which may be of general interest. Firstly, I want to correct a typo. Please consider the data_out port declaration in ROM to be as follows. This port is supposed to be 9 bits, not 8 bits.

```
data_out : out std_logic_vector(8 downto 0)
```

The overall design (let us call it “overall_design”) would consist of the following.

- a) a RAM
- b) a ROM
- c) a MAC
- d) a circuit (let us call it “part_d”) which performs filtering using the above three

The first three are instance of the given components. For creating instances of pre-designed components, see lecture 11 (20th Oct) or refer to the text book (pages 283 and 791). The fourth one you need to design afresh. You can do it in two ways.

- i. Design part_d as a separate component (entity + architecture). Then instantiate all four (RAM, ROM, MAC, part_d) in overall_design, connecting them through signals.
- ii. Instantiate only the first three (RAM, ROM, MAC) in overall_design. Here part_d will not be a separate component, but a bunch of concurrent statements in overall_design.

Note that to compute each pixel of the filter image, part_d needs to perform 9 MAC operations sequentially and then store the result in RAM. Before doing a MAC operation, one RAM read and one ROM read operations would be required. For each operation, part_d has to supply inputs to other components in one cycle and get the result in the next cycle. All this sequencing has to be done by an FSM in part_d. There will also be a bunch of counters that will help in producing addresses for RAM and ROM. These counters need not be separate components, these could be signals local to part_d.

This assignment does not involve design of a complete system. To keep it simple, input/output parts have been left out (except for a button, a switch and a clock). It is assumed that the image to be filtered and coefficients for both the filters are somehow pre-loaded and the filtered image remains in the memory till next filtering operation. You may store the image and filter matrices row-wise or column-wise. The choice has been left open.