Required Equipment

No FPGA hardware is required for this lab. Simulation only.

Image Preprocessing

First open the image with the .bmp template in matlab, add the salt and pepper noise to it with the add_noise function (below is the code), and save it with the .jpeg template:

Figure 1– MATLAB add_noise function

```
function [ noise_image ] = add_noise( image , black ,
white , range )

%   Detailed explanation goes here

g_image = rgb2gray(image);
imwrite(g_image , '… \grey_image.jpeg');

[ y , x ] = size(g_image);

random_value = randint(y , x , [0 range]);

noise_image = g_image;

for i = 1 : y
    for j = 1 : x

        if ( random_value(i , j) <= black )
            noise_image(i , j) = 0;
        end

            if ( random_value(i , j) >= white )
                noise_image (i , j) = range;
            end

    end
end

nn_image = noise_image';
n_image = nn_image(:)';

figure(1); imshow(noise_image);
imwrite(noise_image , 'C…\noisy_image.jpeg');

nn_image = reshape(n_image , [ 1 , 410*361]);
dlmwrite(…\noisy_image.text' , nn_image);

end
```

In the add_noise function, we first black and white the image using the rgb2gray ready-made function, then create a matrix the size of the image with random numbers between 0 and range (the input parameter of the add_noise function) and place the numbers in this larger matrix. Find white (add_noise function input parameter) and set the numbers in the image matrix to 255; again, find the number of values in the random

matrix less than black (add_noise function input parameter) and put 0 in the image matrix in this place. Then we make the new image matrix one-dimensional and save it in the .text noisy_image file.
In modelsim, process the images with the input ports: image_input [7: 0], enable, enable_process, and clk and the output ports: image_output [7: 0] and finish. Furthermore, give the module the parameters Width and Depth, which are the length and width of the image, and filter_size, which is the size of each dimension of the processing window (filter_total_size = filter_size × filter_size). Once the enable is started, receive the data of the input image in each clock, and in one register, store all the bytes of the input image, and when all are received, if enable_process is on, start processing the image.
In the test module, first put the noisy_image.text file containing the noisy image data in a register. Afterward, make enable on and receive the registered input. Then, make enable=0 and enable_process=1 to perform all data processing. Next, receive the filtered image data. At the end, put the filtered image data in the filtered_image.text file.
Read the image using matlab. First, read the filtered_image.text file with the **dlmread** ready function, adjust its dimensions with the **reshape** function, and display the filtered image with **imshow**.

Image Processing Filters

Median filter

Implement median filtering which is a nonlinear method used to remove noise from images. It is widely used as it is very effective at removing noise while preserving edges. It is particularly effective at removing 'salt and pepper' type noise. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels. The pattern of neighbours is called the "window" (filter size), which slides, pixel by pixel, over the entire image. The median is calculated by first sorting all the pixel values from the window into numerical order, and then replacing the pixel being considered with the middle (median) pixel value.

Low-pass filter (Blurring)

Implement low-pass filter is the basis for most smoothing methods. Using a low pass filter tends to retain the low frequency information within an image while reducing the high frequency information. An image is smoothed by decreasing the disparity between pixel values by averaging nearby pixels. The simplest low-pass filter just calculates the average of a pixel and all of its immediate neighbors. The result replaces the original value of the pixel. The process is repeated for every pixel in the image.

High-pass filter (Blurring)

Implement high-pass filter which can be used to make an image appear sharper. Opposite of the low-pass filter, this filter emphasizes fine details in the image. High-pass filtering works in the same way as low-pass filtering using a different convolution kernel.

Resize

Implement resize filtering. Instead of filter_size, you can have resize_size window and an extra input which determines whether to reduce or increase the size. It is clear that the higher the resize_size, the smaller/larger the image.

Reduce: In each resize window, replace the whole window with a pixel that is the average of the resize window pixels.

Increase: Replace each pixel in the image with a resize window which values are equal to that pixel.

Brightness

Implement brightness module. In this module, there is no need of the filter_size parameter. Two more input ports are required: do_bright and bright[7:0]. The do_bright to specify whether make the image brighter or darker. To this end, add/subtract bright[7:0] to/from each pixel in the image.

Report Requirements

Please submit your project report on CCLE. Include relevant snippets of code into your report. More importantly, show your images before and after noise and filters, and explanations of each filter implementation.