Salt-and-Pepper noise removal based on FPGA

Class name: FPGA structure principle and application

Name: Xingzhou Tang, Qiankun Zhou, Zecheng Xu

Perm number: 20300750006, 20300750028, 20300750010

Instructor: Lingli Wang

Contents

1. Project introduction

This project focuses on the implementation of the median filtering algorithm using FPGA. In the process of digital image processing, noise, especially interference noise and salt-and-pepper noise, is inevitable. The presence of noise severely affects the results of image processing, particularly in edge detection algorithms. The median filtering algorithm is a non-linear smoothing algorithm based on the theory of statistical ranking. It can effectively smooth noise and has a good filtering effect on salt-and-pepper noise, while preserving the edge information of the image. Therefore, it is widely used in edge extraction in digital image processing.

Due to the high requirements for real-time performance and speed in digital image processing, FPGA is a suitable platform for implementing the median filtering algorithm. With the characteristics of pipeline structure and parallelism, FPGA can maximize the speed of the median filtering algorithm and meet the increasing demands for real-time performance and speed in the field of digital image processing.

2. Project Design

2.1 Theory of the project

Median filtering is a common image processing technique used to remove noise from an image. It is a non-linear filtering method that involves sorting the pixel values in the neighborhood of each pixel and selecting the median value as the new value for the current pixel. The experimental process of median filtering is as follows: Firstly, a fixed-size sliding window is defined and placed over each pixel. Then, the pixel values within the window are sorted, and the median value is chosen as the new pixel value for the current pixel. This operation is repeated for every pixel in the image until the entire image is traversed. The final result is the image after median filtering.

The window size for this project is 3x3, and the algorithm for finding the median is as follows: First, sort the data in each column of the window. Find the maximum, median, and minimum values for each column. Next, sort the three maximum values, the three median values, and the three minimum values separately. Then, sort the

minimum value among the three maximum values, the median value among the three median values, and the maximum value among the three minimum values. Finally, the result is the median value obtained from the previous step, which represents the median value of the window. The flowchart for this process is as follows.

The median filtering technique has several advantages. It is effective in removing different types of noise, including salt-and-pepper noise (random occurrence of bright and dark pixels) and other types of noise such as Gaussian noise. By selecting the median value, it can suppress outliers and noise, resulting in a clear image.

Compared to some linear filtering methods like mean filtering, median filtering is able to preserve the edges and fine details of an image while removing noise. It maintains the structure and features of the image while smoothing it, making it a preferred choice in many image processing tasks.

2.2 Flowchart of whole procedure

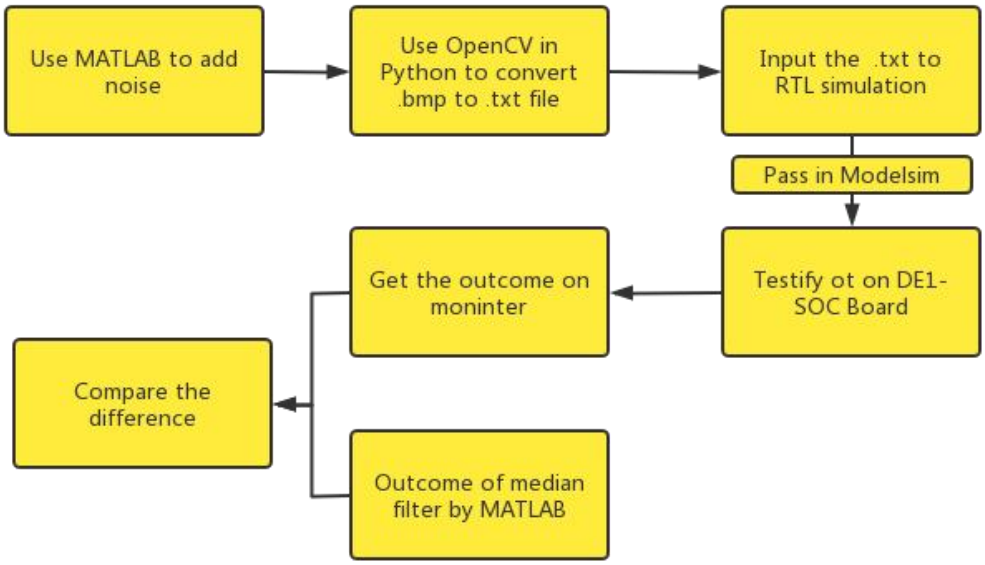


Figure 1: The Flowchart of the whole project

Basically,we first use Matlab to add the Salt-and-pepper noise to the original picture. Then we can use OpenCV, which is a Computer Visual Library created by Intel to convert the .bmp file to .txt in order to further be changed to.mif file.In this case, we can then initialize the memory to store the data of the picture. After write the verilog the code, we can simulate it in ModelSim. We need to revise the verilog code again and again until RTL simulation pass. Then we can finally testify it on the board and compare with the outcome of median filter written in Matlab.

2.3 Verilog system design

2.3.1 main module

The overall flow of the program is illustrated in the following diagram.

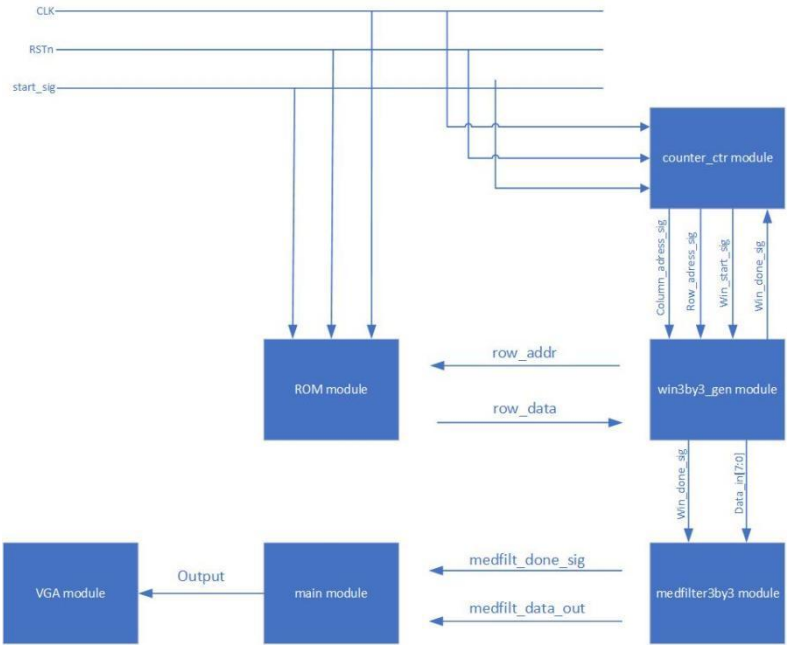


Figure 2: The structure of Verilog file

The main module takes inputs such as the clock signal (clk), reset signal (RSTn), and start signal (Start_sig), and outputs the check signal (Done_sig) as well as the result of the median filtering for the corresponding pixel in that cycle.

The counter module selects the pixel to be filtered in the current cycle and passes its row and column information to the window generation module. The window generation module calculates the addresses of the 9 pixels within the window based on the received pixel position information and sends them to the RAM module.

After the RAM module returns the pixel values within the window, the window generation module sends the values to the median filtering module to perform median filtering. The module then outputs the result of the median filtering for that pixel.

After traversing all the pixels, the result of the median filtering is restored into a .bmp

file, representing the output of the median filtering process.

2.3.2 counter module

The counter module is responsible for determining the pixel selected by the current filtering window. The module takes inputs including the clock signal (CLK), reset signal (RSTn), start signal for filtering (start_sig), and enable signal to detect if the next pixel is valid (nxt_pix_sig).

The outputs of the counter module include the row address (row_addr_sig) and column address (column_addr_sig) of the currently selected pixel within the window. It also provides a detection signal (pix_done_sig) to indicate whether the window has completed the selection of pixels.

When the enable signal is active, the counter module begins its operation. It selects the next pixel in each clock cycle and transfers the row and column data of the selected pixel to the window generation module in the next clock cycle.

2.3.3 RAM module

The RAM module is a memory module that stores the image data. Each pixel's address in the RAM corresponds to its pixel value, similar to a lookup table. The RAM module is typically in the format of a .mif file, which is generated from a .txt format image.

During the simulation in ModelSim, the RAM module receives pixel addresses as input from the window generation module and outputs the corresponding pixel values at those addresses. This helps the window generation module determine the pixel values of each pixel within the window.

2.3.4 window generating module

The window generation module takes as input the row and column of the center pixel in the current window from the counter module. It then outputs the pixel values of the 9 pixels within the window, which serve as the input for the filtering module. This module operates over a total of 11 cycles and utilizes a pipeline concept for data processing.

In each cycle, after obtaining the address of a pixel within the window, the module

sends the address to the RAM module in the next cycle. In the cycle after that, it retrieves the pixel value corresponding to the address from the RAM module. The module continues this process to compute and select the next pixel value within the window, repeating the cycle.

By the eleventh cycle, theoretically, the module will have determined the pixel values of the 9 elements within the window, which serve as the input for the filtering module. However, for pixels located at the edge of the image, this project chooses to set the address of the missing pixel to a latched state. The rationality of this method will be discussed in Chapter 4.

2.3.5 medfilter module

The filtering module uses a 3-level comparison circuit, as shown in the diagram.

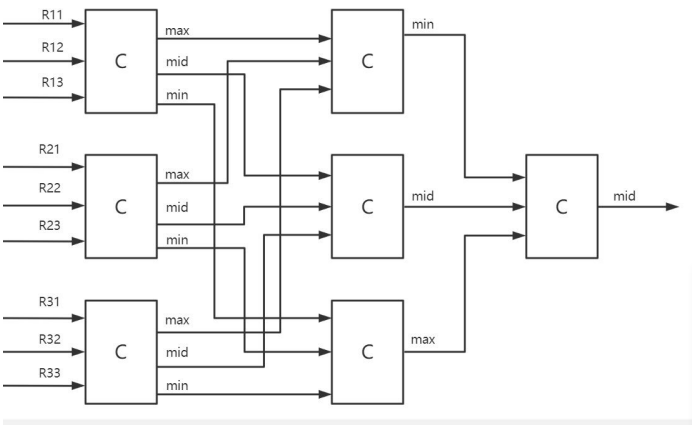


Figure 3: The algorithm of median filter

The first level consists of three three-input comparators, with the output data of each comparator arranged in sequential order. The three smallest values from the three sets of comparison results are grouped together, the three middle values are grouped together, and the three largest values are grouped together for the second round of comparison.

The second level comparison circuit is similar to the first level operation, outputting their maximum values for the three minimum values, outputting their middle values for the three middle values, and outputting their minimum values for the three maximum values. The third round of comparison outputs the middle value of three values.This output is the median of these nine data.

3. Process of the project

3.1 Process outline

This project utilizes several software tools, including MATLAB, Python, Quartus II, and ModelSim. After selecting an image, the first step is to use MATLAB code to convert the color image to grayscale and add salt-and-pepper noise to the original image. Then, Python code is used to convert the .bmp file to a .txt file, which serves as input for the Verilog code.

ModelSim is primarily used to simulate the circuit and verify the correctness of the circuit logic. Quartus II is used for RTL simulation using the DE1 board and generating experimental results.

At the end of the experiment, the output results from Quartus II are compared with the results obtained by applying median filtering to the salt-and-pepper noise using MATLAB. This comparison is done to validate the success of the project.

3.2 procedure of the project

3.2.1 Matlab

Two 512*512 pixels pictures are selected as the materials of the project.

Figure 4: Original picture

After selecting the image, the first step is to use MATLAB to convert the color image

to grayscale. To do this, the imread() function is used to obtain the RGB data of the original image. Then, mathematical formulas are applied to convert the RGB data to YCrCb data, where Y represents the luminance component of each pixel. By replacing all the RGB data with the luminance Y and using the imshow() function, the grayscale image can be obtained.s

Next, MATLAB's built-in function imnoise() is used to add salt-and-pepper noise to the original image, resulting in a .bmp file that serves as the input for the filtering process in this experiment.

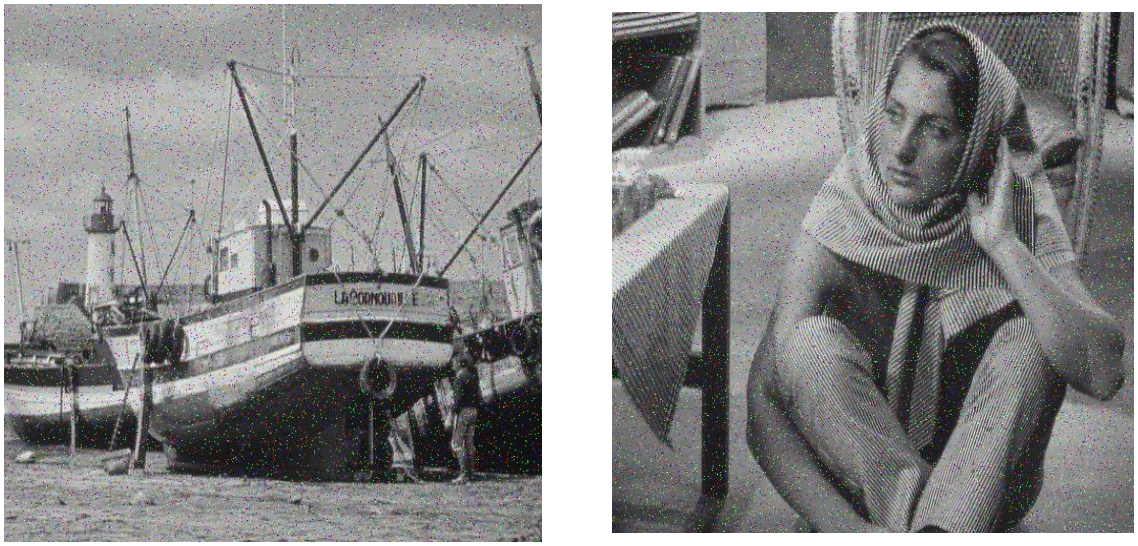
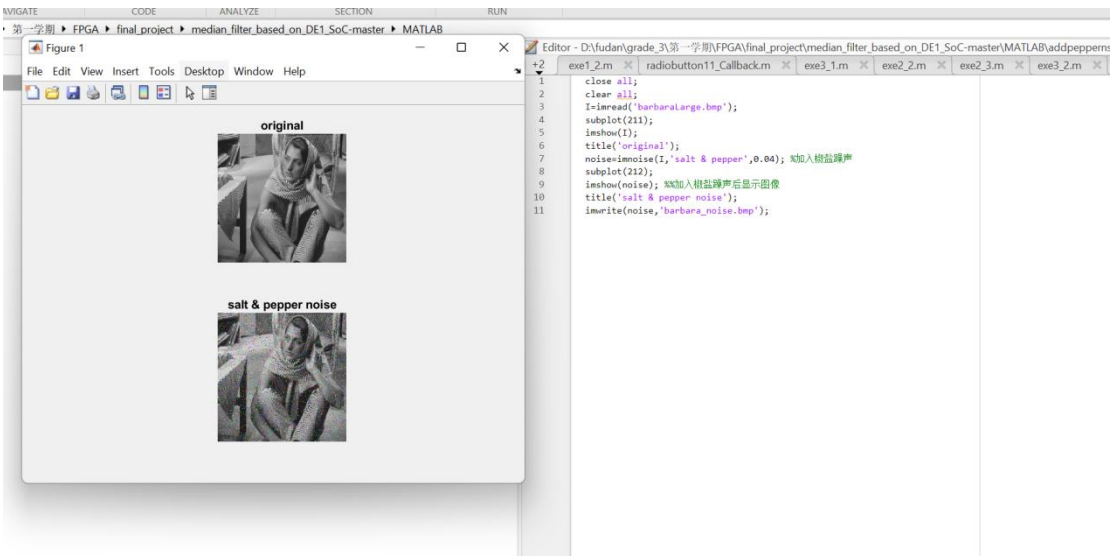
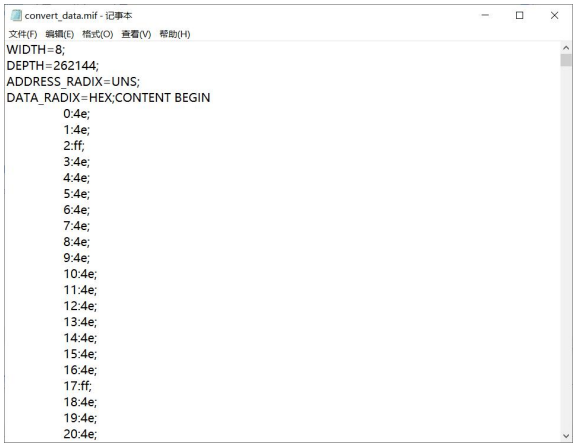


Figure 5 : Picture with Salt-and-pepper noise



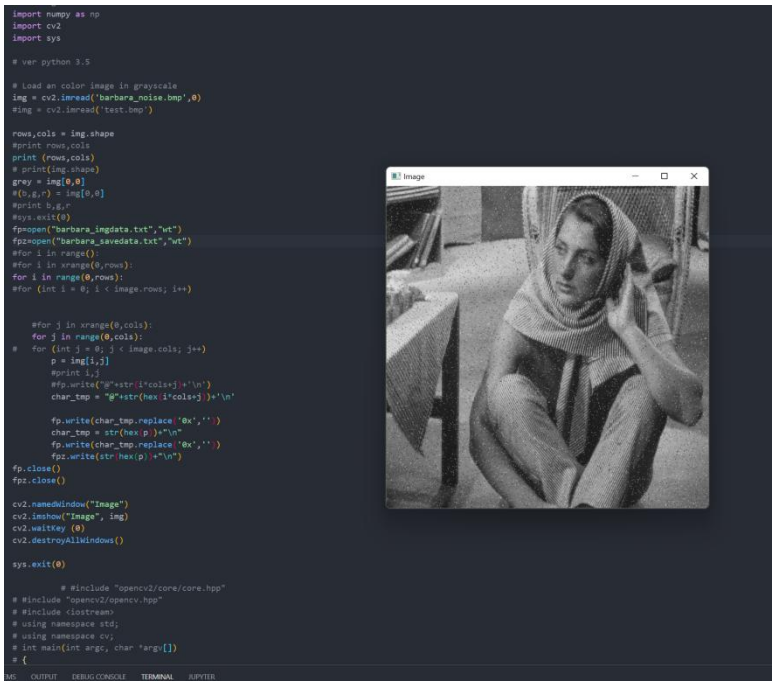
And this is the pictures after adding the salt-and pepper noise.

Additionally, by using the OpenCV library in Python, we can convert the .bmp file to a .mif file, which is used to configure the RAM in Quartus II for simulation.



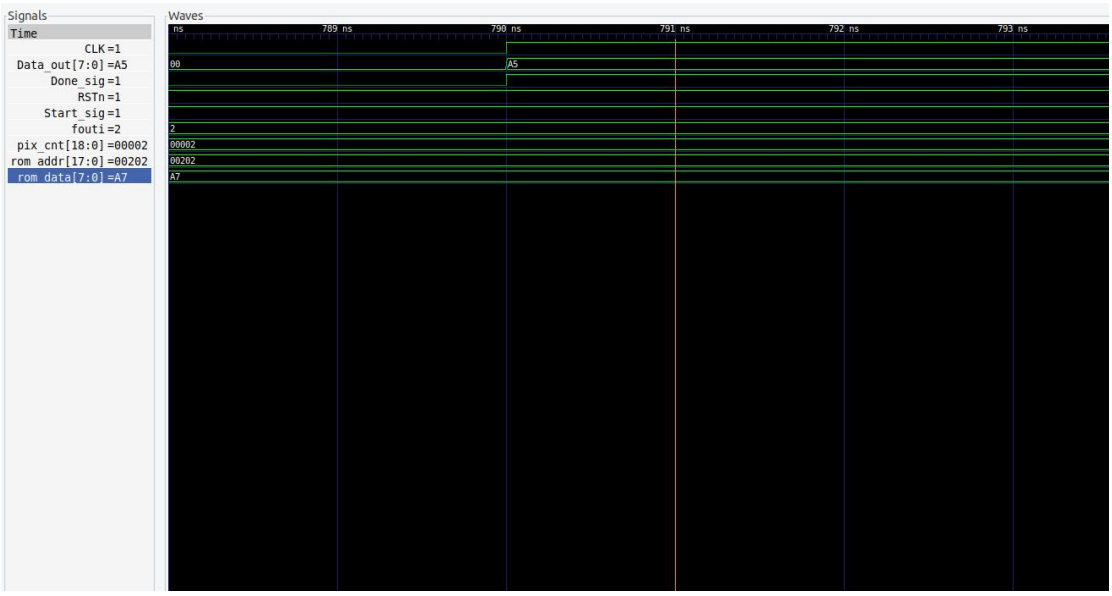
3.2.2 Python

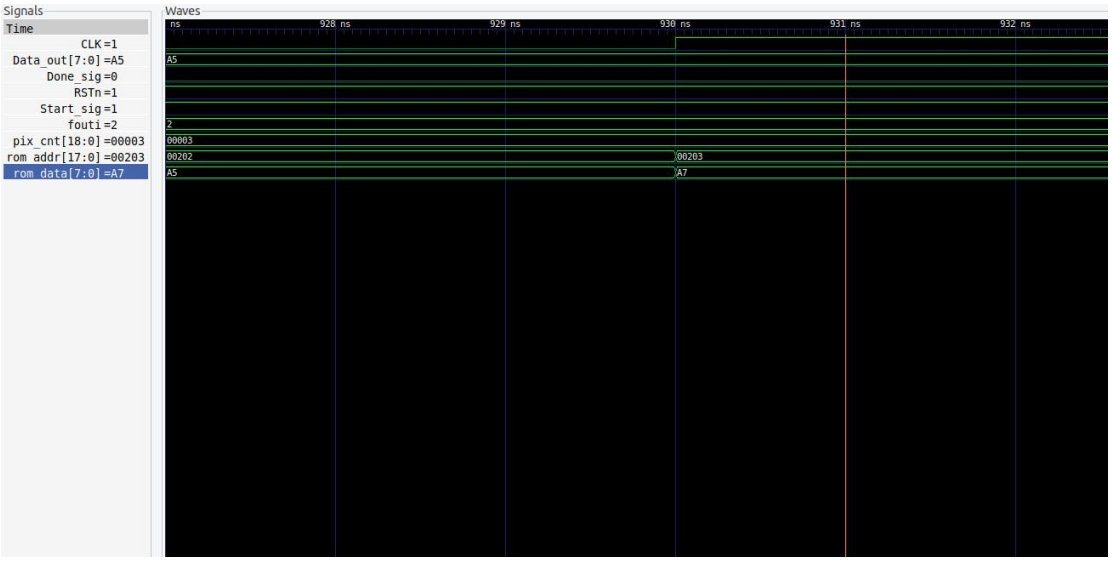
Python is used to convert pictures in .bmp format to .txt files as the input of verilog while using ModelSim to simulate.



3.2.3 Modelsim RTL simulation

When performing circuit-level simulation using ModelSim, with the input being a .txt file generated from the image, the final simulation results are as follows.





It can be observed that different pixels are selected in each clock cycle, indicated by the varying rom_addr, and the output filtered results data_out are also different. After applying median filtering to all the pixels, the filtered .txt file is converted back to a .bmp file using MATLAB, resulting in the simulated output of the Verilog code.

编辑器 - D:\MATLAB\bin\transfer.m

transfer.m

```
% Read the text file
data = dlmread('test.txt');
% Reshape the data into a 512x512 matrix
image_data = reshape(data, 512, 512);
% Scale the data to the range 0-255
image_data = mat2gray(image_data) * 255;
% Create a blank BMP image
bmp_image = zeros(512, 512, 'uint8');
% Assign the image data to the BMP image
bmp_image(:) = image_data(:);
% Save the BMP image
imwrite(bmp_image, 'output2.bmp');
```

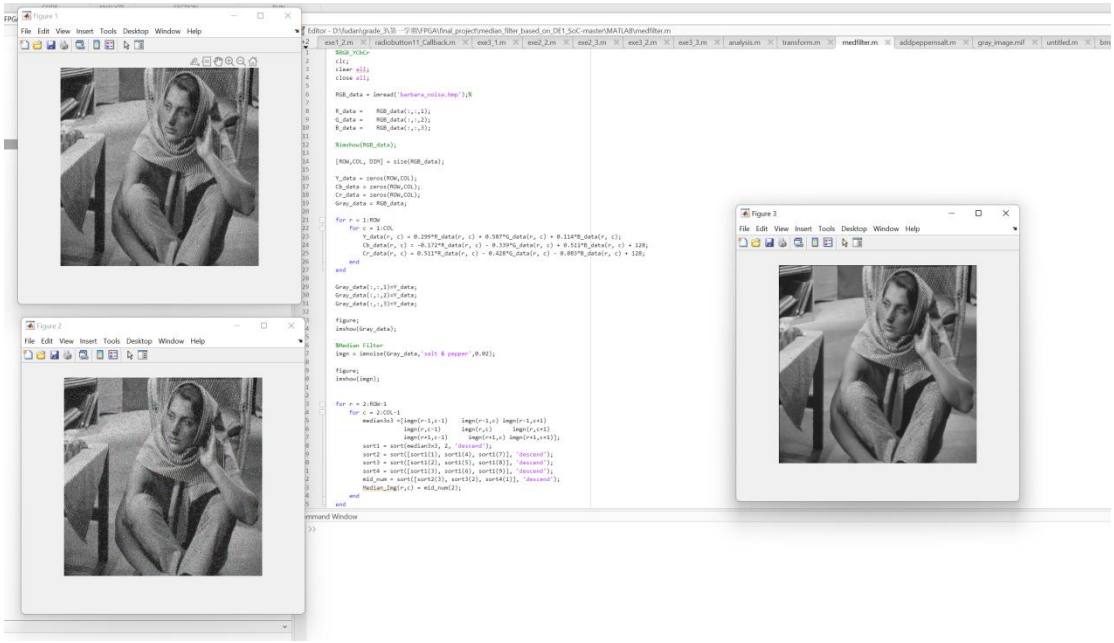
The obtained simulation results show that the filtered image closely resembles the original image, and the salt-and-pepper noise is mostly removed, providing preliminary validation of the correctness of the Verilog code.

3.2.4 Quartus ii simulation

Lastly, Quartus II is used for simulation on the DE1 board. Since ROM is a read-only memory, the content of the image needs to be converted into a .mif file using MATLAB and added to the configuration file for Quartus II simulation. The simulation results are stored in the .mif file, and by converting the .mif file back to a .bmp file, the final result of the median filtering can be obtained.

3.2.5 Result comparison

Using MATLAB can also apply median filtering to images. The process is as follows.



The results of median filtering using Quartus II and MATLAB are shown below, with the Quartus II result on the left and the MATLAB result on the right.

Figure : Outcome of the board(left), Outcome of Matlab(right)

Figure : Outcome of the board(left), Outcome of Matlab(right)

It can be observed that both effectively removing the impact of salt-and-pepper noise on the image without causing significant distortion. Of course, there is still a slight difference between the outcome of MATLAB and DE! Board. We find that the outcome of board is a little bit brighter than . We conclude that that is because MATLAB have many well-designed built-in library inside, which help optimize the outcome. That why the outcome of the MATLAB still more similar to the original one.

4. Problems and solutions

4.1 Edge pixels problem

When implementing the window generation module, there are no complete nine pixels in the boundary pixels. At first, we attempted to connect the pixel points in a spherical shape, that is, the previous line of the first row of pixel points is the 512nd line. However, this needs to be discussed in at least 8 cases, and the code complexity will increase. Later, this method was adopted: for boundary pixels, if one of their nine pixels does not exist, the address register will be locked and remain unchanged. This will inevitably lead to deviation in the output results of boundary pixels, but an image consists of 512 * 512 pixel points, and boundary pixels only account for a small part, resulting in no problem in the image results.

4.2 .txt input problem

When using verilog for filtering, approximately 500 output data are x. After comparing the input and output, it was found that the number of units in the input data (hexadecimal, i.e. numbers less than 16), such as 0x3, was read as x3 by verilog instead of 03. Using regular expressions, add '0' before these single digits as input, and the output data no longer contains x.

5. Conclusion

In this lab, we basically write a MATLAB Script to generate the Salt-and-Pepper noise on the target .bmp file and change the added noise one to txt format in order to input it into the DE1 SOC RAM. Then we can change the Rom through Memory Initialization File(.mif). And download .sof file to get the result from FPGA and compare the outcome with MATLAB.

This kind of Salt-and-Pepper is pretty common in picture. We first decide median filter is more desirable rather than moving-average filter. Some unexpected problems occur(pixel problem, input problems .etc). Fortunately, the problems are finally solved decently. The final filtered outcome of the result based on FPGA is pretty similar with the outcome of MATLAB.

This project involves not only hardware design by verilog but also some software (e.g. python and MATLAB script) used to to digital signal processing procedure. Our team members all learn a lot through this project.