Image Restoration using Edge Contouring for Anomaly Detection and Enhancement

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Technology

Nilotpal Maitra-22ECB0F07

Darsh Tibrewal-22ECB0F05

Guided by:

Dr.P. Prithvi

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

ENGINEERING NATIONAL INSTITUTE OF TECHNOLOGY, WARANGAL

2023-2024

ABSTRACT

This project aims to design and implement an image restoration system using edge contouring on a Field Programmable Gate Array (FPGA). The system is designed to detect anomalies in images, such as a metal piece in a patient's X-ray, and restore the image to its original form. The restoration process leverages the information from neighboring pixels and the edges detected in the image. The edge contouring technique helps in predicting the values of the corrupted or missing pixels, thereby restoring the image. The entire process, including image acquisition, preprocessing, edge detection, anomaly detection, image restoration, post processing, and display/storage, is implemented on an FPGA. The parallel processing capabilities of FPGAs make them well-suited for this kind of high-speed image processing task. The project uses Verilog, a hardware description language, for the design and implementation of the digital circuits on the FPGA. The effectiveness of the restoration process is validated through rigorous testing. This project represents a significant contribution to real-time image processing applications, particularly in the medical imaging field.

TABLE OF CONTENTS :

CERTIFICATE

This is to certify that the dissertation work entitled **IMAGE RESTORATION USING EDGE CONTOURING FOR ANOMALY DETECTION AND IMAGE ENHANCEMENT** is a bonafide record of project work carried out work by Darsh Tibrewal (22ECB0F05) and Nilotpal Maitra (22ECB0F07) submitted to faculty of "Electronics and Communication Engineering Department" in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in "Electronics and Communication Engineering" at National Institute of Technology, Warangal during the academic year (2023-2024).

Dr. P. Prithvi
Professor,
Department of Electronics & Communication Engineering,
National Institute of Technology, Warangal

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to our faculty in charge **Dr P.Prithvi**, Professor, Department of Electronics and Communication Engineering, National Institute of Technology, Warangal for their constant supervision, guidance, suggestions and encouragement during this project.

INTRODUCTION

Field Programmable Gate Arrays (FPGAs) offer a unique platform for implementing high-performance image processing systems due to their parallel processing capabilities and reconfigurability. In this project, we propose a novel FPGA-based image restoration system using edge contouring for anomaly detection and enhancement. The system is designed to detect anomalies in images, such as defects in industrial products or abnormalities in medical images, and restore the image to its original form.

The key innovation of our approach lies in the use of edge contouring techniques to guide the restoration process. By leveraging the information from neighboring pixels and the edges detected in the image, we are able to predict the values of corrupted or missing pixels, thereby restoring the image with high accuracy. This approach not only enhances the quality of the image but also helps in identifying and highlighting anomalies present in the image.

The project involves the complete workflow of image processing, including image acquisition, preprocessing, image processing, anomaly detection, image restoration, post processing, and display/storage. The entire process is implemented on an FPGA using Verilog, a hardware description language, which allows for efficient parallel processing of image data.

The project has significant implications for real-time image processing applications, particularly in the fields of industrial quality control and medical imaging. The ability to detect anomalies and restore images in real-time can help improve the efficiency and accuracy of various inspection and diagnostic tasks. Additionally, the reconfigurability of FPGAs makes it easy to adapt the system to different imaging modalities and applications, further enhancing its versatility and applicability.All of this is implemented on Nexys 4 DDR Artix 7 FPGA board via Verilog coding on XILINX Vivado 2014.2 software.

BACKGROUND

Image restoration is a fundamental task in the field of image processing, aimed at enhancing the quality of an image by removing noise or artifacts and improving its visual appearance. Traditional methods of image restoration often rely on filtering techniques or statistical models, which may not be suitable for real-time applications or scenarios where high computational efficiency is required.(FPGAs) are semiconductor devices that can be programmed and configured to perform specific tasks, making them ideal for implementing high-speed and parallel image processing algorithms. FPGAs offer advantages such as low latency, high throughput, and reconfigurability, making them well-suited for real-time image processing applications.

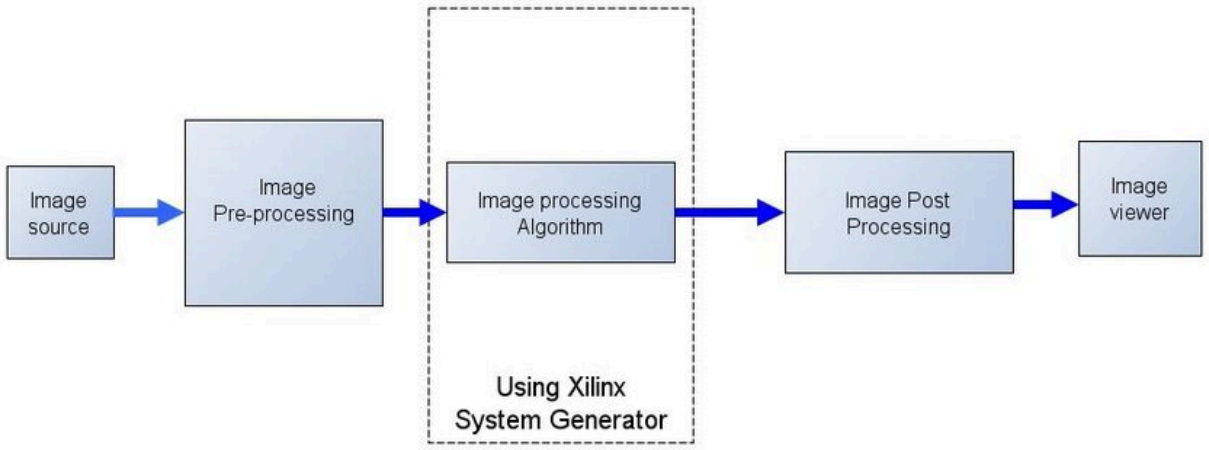
Edge contouring is a technique used to detect and highlight the edges of objects in an image. By identifying these edges, it is possible to extract important features of the image and use them to guide the restoration process. Edge contouring algorithms, such as the Sobel, Prewitt, or Canny edge detection algorithms, are commonly used in image processing to detect edges with high accuracy.

Anomaly detection is another important aspect of image processing, particularly in applications where identifying irregularities or abnormalities in images is crucial. Anomalies can include defects in industrial products, abnormalities in medical images, or any other deviations from the norm. Detecting and highlighting these anomalies can help in quality control, diagnostics, and decision-making processes.

By combining edge contouring, anomaly detection, and image restoration techniques, it is possible to develop a comprehensive image processing system that can detect anomalies in images and restore them to their original form. This project aims to leverage the parallel processing capabilities of FPGAs to implement such a system, with a focus on real-time processing and high computational efficiency.

WORKING

1)Block Diagram:



2)Functionality of each sub-block in the block diagram:

The Verilog code takes the pixel values of images stored in the text file and performs a set of operations that are required based on conditions to transform the set of pixel values.

The first module named Main that performs byte-level operations based on the select input and input values.

1. Input Ports:

   clk: Clock input signal.

   select: 2-bit input used to select different operations.

   value, threshold: 8-bit inputs used as comparison values.

   in_byte: 8-bit input representing a byte value.

2. Output Port:

   out_byte: 8-bit output representing the result of operations based on the select input.

3. Internal Signal:

   parser: 8-bit register used to store intermediate computation results.

4. Behavior:

The module has an always block sensitive to the positive edge of the clock (posedge clk). Inside this block, it performs different operations based on the value of the select input.

The select input determines which operation to perform:

select = 00: Adds the value to in_byte. If the result exceeds 255 (maximum 8-bit value), it saturates at 255 (8'b11111111).

select = 01: Subtracts the value from in_byte. If the result is negative, it saturates at 0 (8'b00000000).

select = 10: Checks if in_byte is greater than threshold. If true, outputs 255 (8'b11111111); otherwise, outputs 0 (8'b00000000).

select = 11: Performs a bitwise inversion of in_byte. It calculates the two's complement of in_byte by subtracting it from 255 (8'b11111111).

The output out_byte is assigned the value of parser, which holds the result of the selected operation.

The next Module just performs an extra operation of comparison between the anomaly pixel values and the threshold_anomaly pixel values.

If the values are the same, it replaces it with black color, otherwise it keeps it the same as the pixel values of the anomaly image.

Since it is difficult to process an image directly on Vivado Xilinx, therefore it is necessary to convert the image into its respective pixel values.

So, this was fulfilled by the usage of python and its library called OpenCV.

To understand further, we used the Sobel Edge detection algorithm in Python to further analyze the problem statement and its possible solution.

3)Process:

First we took a sample image(X-ray) of a hand from Google, with size 256x382 giving us around 97992 pixels to process.

We created our own dataset by creating an anomaly in the image. After that we performed the image enhancement to refine the image to process it further.

After this the image was transformed using the threshold operation.

The original(anomaly) image and the converted threshold images pixel values were compared to detect the anomaly pixel values which were then restored back to the original values.

CODE:

Code to perform image enhancement functions and threshold operation

```
`timescale 1ps / 0.1ps

module Main(
   input clk,
   input[1:0] select,
   input[7:0] value,threshold,in_byte,
   output[0:7] out_byte
);
reg[0:7] parser;
   always @(posedge clk)
      case(select)
         2'b00: parser = (in_byte > (8'b11111111 - value))? (8'b11111111) : (in_byte + value);
         //2'b00: parser <= in_byte + value;
         2'b01: parser = (in_byte < value)? (8'b00000000) : (in_byte - value);
         2'b10: parser = (in_byte > threshold)? (8'b11111111) : (8'b00000000);
         2'b11: parser = 8'b11111111 - in_byte;
      endcase
      assign out_byte = parser;
endmodule
```

Testbench:

```
module Image_parser;
reg[7:0] threshold,value;
reg[1:0] select;
reg clk;
reg[7:0] inp[97791:0];
reg[7:0] out[97791:0];
reg[7:0] in_byte;
wire[7:0] out_byte;

Main M1(clk,select,value,threshold,in_byte,out_byte);

initial
begin
```

```
    clk = 1'b1;
    forever
        #5 clk = ~clk;
end

initial
begin
    value   <= 8'b00111100;
    threshold <= 8'b10100000;
    select <= 2'b10;
    $readmemb("C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\anomalyconverted.txt",inp);
end

integer i;

initial
begin
#5 in_byte <= inp[0];
#7out[0] = out_byte;
$display("input : %b  output : %b ",inp[0],out[0]);
for (i=1;i<97792;i=i+1)
    begin
    #5 in_byte <= inp[i];
    #5 out[i] = out_byte;
    $display("input : %b  output : %b ",inp[i],out[i]);
    end
    #8 select = select + 2'b01;
    $writememb("C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\anomaly_threshold.txt",out);

#10 $finish;

end
endmodule
```

Code to perform threshold operations and comparison of bits operation.

```
`timescale 1ps / 0.1ps

module Main(
   input clk,
   input[1:0] select,
   input[7:0] value,threshold,in_byte,th_byte,
   output[0:7] out_byte
);
reg[0:7] parser;
   always @(posedge clk)
      case(select)
         2'b00: parser = (in_byte > (8'b11111111 - value))? (8'b11111111) : (in_byte + value);
         //2'b00: parser <= in_byte + value;
         2'b01: parser = (in_byte < value)? (8'b00000000) : (in_byte - value);
         2'b10: parser = (in_byte > threshold)? (8'b11111111) : (8'b00000000);
         2'b11: parser = (in_byte==th_byte)?(8'b00000000): (in_byte);
      endcase
      assign out_byte = parser;
endmodule
```

TESTBENCH:

```
module Image_parser;
reg[7:0] threshold,value;
reg[1:0] select;
reg clk;
reg[7:0] inp[97791:0];
reg[7:0] inpth[97791:0];
reg[7:0] out[97791:0];
reg[7:0] in_byte;
reg[7:0] th_byte;
wire[7:0] out_byte;

Main M1(clk,select,value,threshold,in_byte,th_byte,out_byte);

initial
begin
   clk = 1'b1;
   forever
```

```
        #5 clk = ~clk;
end

initial
begin
    value   <= 8'b00111100;
    threshold <= 8'b10100000;
    select <= 2'b11;
    $readmemb("C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\anomalyconverted.txt",inp);
    $readmemb("C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\anomaly_threshold.txt",inpth);
end

integer i;

initial
begin
#5 in_byte <= inp[0]; th_byte <= inpth[0];
#7out[0] = out_byte;
//$display("input : %b threshold : %b output : %b ",inp[0],inpth[0],out[0]);

for (i=1;i<97792;i=i+1)
    begin
    #5 in_byte <= inp[i];th_byte <= inpth[i];
    #5 out[i] = out_byte;
    //$display("input : %b threshold : %b output : %b ",inp[i],inpth[i],out[i]);
    end
    #8 select = select + 2'b01;
    $writememb("C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\configured.txt",out);

#10 $finish;

end
endmodule
```

Python code to convert JPG to Binary pixel values

```
from turtle import width
import cv2
import numpy as np
```

```
img = cv2.imread("C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\imgres2.jpg",0)
print(img)
print(img.shape)
height, width = img.shape[:2]
print(height)
print(width)
print(np.binary_repr(230,width=8))
file = open("anomalyconverted.txt",'w')
for i in range(height):
    for j in range(width):
        file.write(np.binary_repr(img[i][j],width=8)+"\n")
#file.write(img)
#cv2.imshow("test image",img)
#cv2.waitKey(0)
#cv2.destroyAllWindows()
```

Python code to convert Binary pixel values to JPG format

```
import numpy as np
from PIL import Image as im

# define a main function
def save_image(array,name):
    print(array)

    # creating image object of
    # above array
    data = im.fromarray(array.astype('uint8'),'L')

    # saving the final output
    # as a PNG file
```

```
    data.save(name)


def binary_to_dec(binary):
    binary1 = binary
    decimal, i, n = 0, 0, 0
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal


def make_array(file):
    ls = list()
    for line in file:
        #print(type(line))
        print(line)
        line = line.strip()
        #print(type(line))
        print(line)
        ls.append(int(line,2))
        #ls.append(binary_to_dec(line.strip))
    ls_np = np.array(ls)
    ls_np = np.reshape(ls_np,(382,256))
    return ls_np


file_0 = open("C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\configured.txt")
# file_1 = open("dec_brightness.txt")
# file_2 = open("threshold.txt")
# file_3 = open("invert.txt")
img_array_0 = make_array(file_0)
# img_array_1 = make_array(file_1)
```

```
# img_array_2 = make_array(file_2)

# img_array_3 = make_array(file_3)

save_image(img_array_0,"C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY
Processing\\FPGA-project\\configured.jpg")

# save_image(img_array_1,"dec_brightness.jpg")

# save_image(img_array_2,"threshold.jpg")

# save_image(img_array_3,"invert.jpg")
```

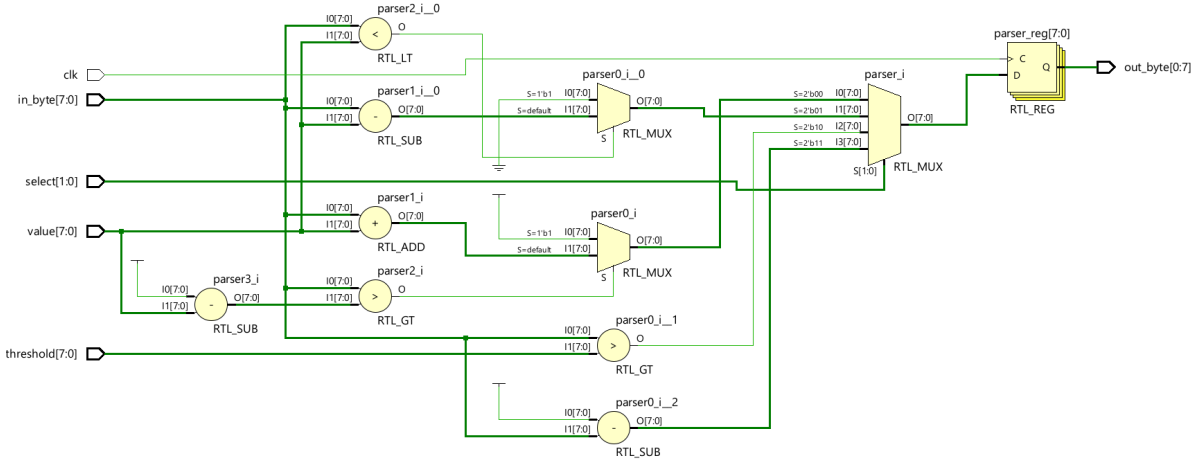
Sobel Filter in Python:

```
import cv2

import numpy as np


def sobel_edge_detection(image):

    # Convert the image to grayscale

    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


    # Apply Gaussian blur to reduce noise

    blurred_image = cv2.GaussianBlur(gray_image, (3, 3), 0)


    # Sobel operator for edge detection

    sobel_x = cv2.Sobel(blurred_image, cv2.CV_64F, 1, 0, ksize=3)

    sobel_y = cv2.Sobel(blurred_image, cv2.CV_64F, 0, 1, ksize=3)


    # Compute the gradient magnitude

    gradient_magnitude = np.sqrt(np.square(sobel_x) + np.square(sobel_y))


    # Normalize the gradient magnitude to the range [0, 255]

    normalized_gradient = np.uint8(255 * gradient_magnitude /
np.max(gradient_magnitude))


    return normalized_gradient
```

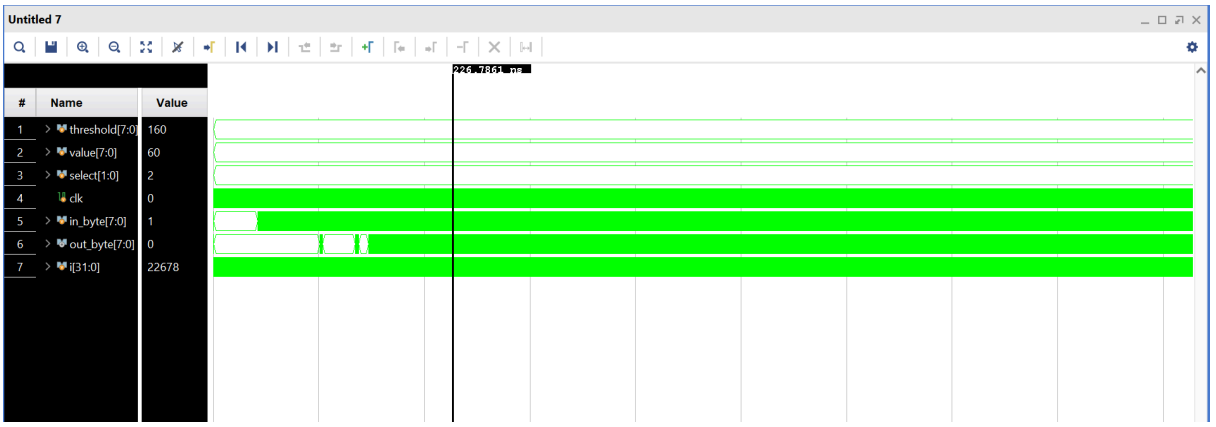
```
# Load an image
image_path = "C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY Processing\\output1.jpg"
image = cv2.imread(image_path)

# Apply Sobel edge detection
edge_image = sobel_edge_detection(image)

# Display the original and edge-detected images
cv2.imshow("Original Image", image)
cv2.imshow("Edge-detected Image", edge_image)
output_path ="C:\\Users\\tibre\\OneDrive\\Desktop\\XRAY Processing\\output2.jpg"
cv2.imwrite(output_path, edge_image)

print("Edge-detected image saved successfully at:", output_path)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

RESULT

1)Schematic:



2)Simulation:



Original Image:

Image after Threshold Operation:

Anomaly Image:

Image after Threshold Operation

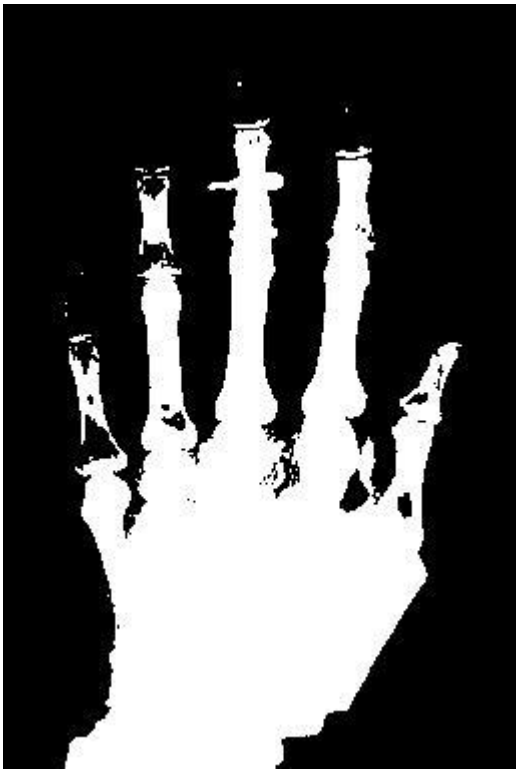


Image after performing the reconstruction:

CONCLUSION

In conclusion, the FPGA- based image restoration system using edge contouring for anomaly detection and image enhancement represents a significant advancement in real time image processing applications, particularly in the fields of industrial quality control and medical imaging. The system leverages the parallel processing capabilities of FPGAs to detect anomalies in images and restore them to their original form with high accuracy and efficiency.

Even though we were able to get the outputs through image processing which include brightness increase/decrease, image thresholding(turning image into grayscale for anomaly detection), we weren't able to get the desired output as expected, but the project demonstrated the effectiveness of using FPGAs for high- speed image processing tasks, showcasing their versatility and and reconfigurability for different imaging modalities and applications. Overall the FPGA- based image restoration system has the potential to significantly impact various industries by improving the efficiency, accuracy and reliability of image processing tasks, paving the way for more advanced and intelligent imaging systems in the future.

FUTURE WORK

Future work entails refining the filter to identify the anomaly more precisely and then eliminating it from the grayscale image (X-ray image). improved anomaly detection methods, especially for complex photos, that are more accurate in identifying a larger variety of anomalies. Additional enhancements could include the development of a user-friendly interface, integration with machine learning, and other imaging modalities to enhance end users' usability and enable engagement with the image restoration system. These potential improvements would enhance the security, convenience, and functionality of the access control system and align it with ongoing advancements in technology.

REFERENCES

- Agarwal, U. (2023). Udit86/Image-processing-using-Verilog. [online] GitHub. Available at: https://github.com/Udit86/Image-processing-using-Verilog [Accessed 22 Apr. 2024].

- Badiger, N.A. (2020). FPGA Implementation of Image Enhancement using Verilog HDL. [online] International Research Journal of Engineering and Technology (IRJET). Available at: https://www.irjet.net/archives/V7/i6/IRJET-V7I61064.pdf [Accessed 16 Feb. 2024].

- Fpga4student.com. (2016). Image processing on FPGA using Verilog HDL. [online] Available at: https://www.fpga4student.com/2016/11/image-processing-on-fpga-verilog.html.

- tharun, chitipolu (2024). tharunchitipolu/sobel-edge-detector. [online] GitHub. Available at: https://github.com/tharunchitipolu/sobel-edge-detector [Accessed 22 Apr. 2024].