Summer 2021

Implementation Of An Improved Image Enhancement Algorithm On FPGA

Prit Ghanshyambhai Patel

Implementation Of An Improved Image Enhancement Algorithm On FPGA

By

Prit Ghanshyambhai Patel

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2021

Implementation Of An Improved Image Enhancement Algorithm On FPGA

by

Prit Ghanshyambhai Patel

APPROVED BY:

A. Rahimi
Department of Mechanical, Automotive and Materials Engineering

H. Wu
Department of Electrical and Computer Engineering

M. Khalid, Co-Advisor
Department of Electrical and Computer Engineering

A. Ahmadi, Co-Advisor
Department of Electrical and Computer Engineering

May 18th, 2021

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Image processing plays very crucial role in this digital human world and has rapidly evolved with the development of computers, mathematics and the real-life demand of variety of applications in wide range of areas. This wide range of areas includes remote sensing, machine/ robot vision, pattern recognition, medical diagnosis, video processing, military, agriculture, television, etc. Image processing has two important components which are image enhancement and information extraction. Since image enhancement works at the front end with the initial raw inputs, it works like a backbone in image processing. When it comes to implementing these image enhancement techniques and developing applications, these tasks are bit demanding in the choice of processing units because the demand of high resolution. This emerges the necessity of a high speed, powerful and cost-effective processing unit. In this thesis we present an improved image enhancement algorithm in terms of performance and its implementation on FPGA as they satiates the necessity of high speed, powerful and cost-effective processing unit by providing flexibility, parallelization, pipelining and reconfigurability. We have performed a high level synthesis by using MATLAB and implemented an improved image enhancement algorithm on Cyclon V by using Quartus Prime. We have considered an X-ray image size of 1000x1920p for implementation and achieved a decent PSNR values and hardware resource utilization along with the better visual interpretability by our proposed improvements. For achieving a better execution time and power consumption we also offer the task parallelism for the algorithm.

DEDICATION

Dedicated to my father ***Ghanshyambhai Ishvarbhai Patel****, my mother* ***Jayshree Ghanshyambhai Patel*** *and my brother* ***Vedant Patel*** *for their overwhelming enthusiasm, support, love, and for making everything possible.*

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to **Dr. Arash Ahmadi** and **Dr. Mohammed Khalid** for their continuous support and inputs through out my work. I could not have imagined a better mentor for my study. I would also like to thank **Dr. Huapeng Wu** and **Dr. Afshin Rahimi** for their inputs that helped me in refining and improving my work and helping me in guiding my work in proper direction and leading my thesis towards a gracious completion.

Also, I would like to thank my all nearest family members especially my cousins **Dipesh Patel**, **Tirth Patel**, **Rut Patel** and **Akshay Patel**, for keeping me encouraged, without whom I could not have imagined my journey through the University. Special thanks to my teacher **Kuntal Patel** without whom my journey would not be possible.Cheers to my best friends **Pritesh Patel**, **Hetvi Patel**, **Dhruvesh Patel**, **Manan Patel**, and **Vandan Patel** for their constant support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYM

FPGA Field Programmable Gate Arrays

HLS High Level Synthesis

LoG Laplacian of Gaussian

PSNR Peak Signal to Noise Ratio

MATLAB MATrix LABoratory

ASICs Application Specific Integrated circuits

CPU Central Processing Unit

GPU Graphics Processing Unit

USB Universal Serial Bus

PCIe Peripheral Component Interconnect express

LOFAR Low Frequency Array

SKA Square Kilometre Array

HDL Hardware Description Language

SDK Software Development Kit

DNNs Deep Neural Networks

AI Artificial Intelligence

RTL Register Transfer Level

VHDL Very High speed integrated circuit Hardware Description Language

OCR Optical Character Recognition

RAM Random Access Memory

LUTs Look Up Tables

ALU Arithmetic Logic Unit

DSP Digital Signal Processing

RGB Red, Green, Blue

MSE Mean Squared Error

ALM Adaptive Logic Module

I/O Input/Output

CHAPTER 1

Background and Thesis Overview

1.1 Digital Image processing:Overview, Classification and Requirements

No environment is Noise-free and whether it is signal or image, everything needs to be cleaned before its usage to experience a better and optimized output. In other words, there is noise in every system due to various factors such as, sensors, medium, data and so on, and it causes the degradation in the system output as it is directly proportional to the output. Moreover, in these last few decades with the development of the computers, discrete mathematics and the demand of wide range application in different fields from agriculture to medical science and space exploration, the development of digital image processing has been highly affected. Digital image processing is basically a sub-branch of digital signal processing it refers to the improvement in the system output(which is in form of an image) which is very crucial for information extraction and image interpretation[1]. It basically enhances the image quality and provides an ease for human clarification and visualization along with an image data processing for transmission, storage and machine view representation[31]. It is widely useful in remote sensing applications, medical field, machine/robot vision, pattern recognition, video processing, etc[2]. Fundamentally, digital image processing can be classified into five main components which are, acquisition, storage, processing, communication and display. Among these components image processing is a broad subject which consists of different techniques such as Image Restoration, Image Enhancement and information extraction. The focus of this thesis is on the image enhancement since

it is the most important fundamental steps of digital image processing.

1.2 Need for High Level Synthesis

With the increase in the processing power, the trend of the image processing in the device itself rather than using a separate computer has been noticed. For example, in context of the digital cameras, these cameras do not only captures the photos but also performs several image processing operations, communicate the images and extract the data[3]. Although, with the advances of the device technologies, low power general processors struggles in coping up with the high processing demands of the image processing applications because of which the smart devices with FPGAs have started gaining popularity these days. The conventional approach for developing such kind of image processing applications is to develop the test bench of the algorithm and then to port them into hardware description language for FPGA implementation once they function as desired. Thus, the user can develop such kinds of applications by porting the algorithm on FPGA by following the process called High Level Synthesis.

1.3 Motivation Behind The Work

The main advantage of FPGA is its capability to be a standalone quantity which does not require any host computer to run because it has its own input and output. Thus a user can save money and energy on the host. Moreover, FPGA can turn out to be very important by leaving single serial processor and GPUs behind in the following parameters:[27]

1.3.1 Low Latency

Latency plays a crucial role when it comes to real time applications such as autopilot mode of a jet and an airbag system of a car. In these kind of applications, the time period between the system response and the input should be as short as possible.

CPUs having latency smaller than 50 microseconds would be considered as very good but on the other hand, FPGAs are feasible to achieve a latency around or below 1 microsecond. The main reason for this is that FPGAs do not depend on a generic Operating System and they do not communicate via generic buses.

1.3.2 Connectivity and Bandwidth

In FPGA a user can hook up any data source up such as a sensor or a network interface directly to the pins of a chip whereas, in CPUs or GPUs a source has to be connected via a local standardize bus such as USB or PCIe and depend on the Operating System to deliver the data.
This direct connection to the pins provides very high bandwidth which is very useful in some radio astronomy applications such as LOFAR and SKA because of the usage of specialized sensors in them which generates enormous amount of the data.

1.3.3 Energy Efficiency

FPGAs shine in terms of energy efficiency when it comes to logic and fixed point calculations which makes FPGAs more useful in the field of crypto-currency mining. Even Intel is always acclaiming the energy efficiency of FPGA. Although, nowadays ASICs (Application Specific Integrated circuits) are used for this applications since they are built for only one purpose. Though, they require very large upfront investment and a large number of chips to be produced to be cost effective.

1.4 Challenges and Outlined Solution

Now, when it comes to implementing the algorithms the users have different options available and hardware implementation offers a better speed than the software implementation. Though, the development time for the hardware implementation is larger. Most of the designers are familiar with C but the hardware implementation and synthesis require the extensive knowledge of hardware and VHDL or Verilog. Moreover,

there are some tools that offers software to hardware conversion schemes such as Vivado HLS, Altera SDK, MATLAB HDL coder, etc. But again, most of these tools are costly and it requires to restructuring of the code to achieve a better performance. Because of this reason image processing algorithms are very less implemented on a powerful hardware like FPGA even though having a wide range of applications.

- Although the hardware implementation is a bit arduous task, it offers many advantages and provides a better match for these image or video processing algorithms to be implemented. A real time image or video processing algorithm will raise an inconvenience for the single serial processor because it requires images with very large size and high resolution. For example, if we consider a standard 720 pixel video stream at 24 frames per second, it will require at least 66 millions of operations to process that video stream. Moreover, generally most of the image processing algorithms require dozens of the operations on a single pixel. It will turn out be an extreme heavy load for single serial processor.[5] Thus, one can say that FPGAs offer task parallelization which like is a cherry on top.

- Moreover, the Sobel operator which generally is used in the edge detection is very basic and has a performance problem when it is compared to the benchmarks and also it offers very poor signal to noise ratio with the increment in the noise. The overcoming of the performance problem will be done by replacing the Sobel operator with the LoG opeator.

- Apart from that, usage of the Laplacian operator is only limited to the grey-scale images which makes the area of application for the image enhancement algorithm lesser broad[21]. To solve this problem the Local Laplacian filter will be used for the image enhancement algorithm to be useful for coloured images as well.

- Additionally, the algorithm proposed in [17] is sequential and lacks parallelism which can cost more power and execution time when it comes to implementing

this algorithm on any hardware. Moreover, power consumption and execution time are crucial parameters as they are directly proportional to the flow of the algorithm. So, for this we propose a task parallelism and parallel flow of the algorithm.

The aim of this research is to attempt to improve an image enhancement algorithm and implementing a parallel algorithm on a powerful and flexible hardware- FPGA. This approach would not only provide a viable option for implementing image or video processing algorithm without sacrificing the performance parameter like execution time, chip area and power consumption, but it would improve the execution time and power consumption and allow us to witness the capability of FPGA to be a standalone quantity in the applications.

We tackle the problem with the points stated above which we discuss comprehensively in subsequent chapters of this thesis. To summarize the problem statements and solutions for those problems, we show the table below.

Problems	**Proposed Solutions**
Performance problem of Sobel operator	Usage of LoG
Limitations of Laplacian and scope of the algorithm	Usage of Local Laplacian
Sequential nature of the algorithm	Proposal for a task parallelism
Long development time for hardware implementation	Usage of HLS

Table 1.4.1: Problem statements and their proposed solutions

1.5 Thesis Outline

The subsequent chapters of the thesis are organized as follows. In **Chapter 2**, we present the readers with succinct knowledge which lays a technical foundation for the rest of the thesis. As for **Chapter 3**, we present the Literature Survey of related work done in the area. **Chapter 4** explains the implemented Methodology to solve the

problem. **Chapter 5** includes the experiment results and analysis of the proposed solution and comparison with other methods. And lastly, In **Chapter 6**, we conclude the thesis and address the work that can be possibly done in the future to refine the proposed solution.

CHAPTER 2

Overview of Related Technologies

2.1 Image Enhancement

The motive of Image Enhancement is to amplify or sharpen the details or the features of an image so that an image can be used further in the other applications. It makes an image more appropriate by exploring the hidden details of the image due to the various environmental and hardware factors and make the extraction of the contained details more meaningful. Mainly, Image Enhancement techniques can be classified into two broad categories based on the processing level, Spatial Domain Image Enhancement and Frequency Domain Image Enhancement.

2.1.1 Spatial Domain Image Enhancement Techniques

To begin with the processing level, Spatial Domain Image Enhancement Techniques operate at the pixel level and deal directly with the pixel values. These methods refer more efficient computations as well as they require less processing resources for implementation. Here, the pixel values are manipulated to achieve the desired result. In Spatial Domain Image Enhancement Techniques, the image processing function can be expressed as,

$$g(x, y) = T(f(x, y)) \tag{1}$$

Where, f(x,y) : input image, g(x,y) : processed image, T() : operator on f defined over some neighbourhood N(x,y)

Spatial Domain Image Enhancement Techniques directly alter the pixel values or gray levels of the pixels and hence directly affect the overall image uniformly. Here, it is not possible to alter information of the selective parts of the image. There are two main approaches in this methods, point processing and spatial filtering.

2.1.1.1 Point Processing

Here, the enhancement of the image depends only on the gray level of one point, i.e., pixel. According to the equation (1) T refers to the gray level transformation of the image so that point processing is also known as Gray Level Transformation. point processing operations take form of,

$$s = T(r) \tag{2}$$

Where, S refers to the processed image pixel values and T refers to the original image pixel values. Mainly there are four types of Gray Level Transformation; Linear, Logarithmic, Power-law and Piecewise-Linear.

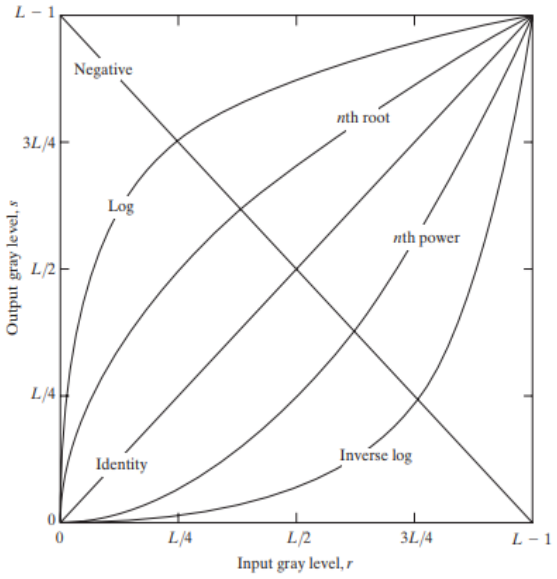

Fig. 2.1.1: Gray-level Transformation graph [29]

2.1.1.1.1 Linear Transformation: Linear Transformation has two categories; Negative Transformation and Identity Transformation.

Negative Transformation: Images as a sets of pixels are digitally stored in form of digital values. A pixel of a gray-scale image has two values; 0 and 1. 1 refers to colour white and 0 refers to the colour gray(or black). The most common pixel format, byte image has the pixel value in form of 8-bit integer from 0 to 255. Here, 0 is considered as black and 255 is considered to be white. The values between them are considered to be the different shades of gray.Now, when negative transformation is applied to the image, the brightest areas are transformed into the darkest areas and the darkest areas are converted into the brightest areas. The negative transformation can be defined as,

$$s = L - 1 - r \tag{3}$$

Where, L-1 is maximum pixel value and r is the targeted pixel value.

Identity Transformation: This type of transformation is used when a user wants identical input and output. Here, each value of the input image is directly mapped to each other values of the output images which results in the same or identical output.

2.1.1.1.2 Logarithmic Transform: This method can be used for the low light images since it expands the values of the dark pixel and compresses the value of the bright pixel. It maps the narrow range of the low gray level intensities to wide range of output values. This transformation can be defined as,

$$s = C * log(1 + r) \tag{4}$$

Where r and s are input pixel value and output pixel value respectively. C is scaling constant. Inverse-log Transform is the opposite to the Log Transform.

2.1.1.1.3 Power-Law Transformation It is also known as Gamma Correction or Gamma Transformation and it is used in enhancing images for different types of display devices since the gamma value of different display devices is different. Different display devices project different intensity levels and clarity according to the built in gamma correction inside them have different ranges. This transformation can be defined as,

$$s = C * r^{\gamma} \tag{5}$$

Where, s is output pixel value,r is input pixel value and γ is a gamma value of a pixel. Here, the variations in γ causes the variation in the enhancement of an image.

2.1.1.1.4 Piecewise-Linear Transformation: Piecewise-Linear Transformation is arbitrary user defined transform which can be categorized in three sub-categories; Intensity Level Slicing, Contrast Stretching and Bit-Plane Slicing.

Intensity Level Slicing: It highlights the specific intensity level of an image as desired. Intensity Level Slicing can emphasize a group level intensity by diminishing all others or vice versa.

Contrast Stretching: Contrast Stretching basically works on entire image and enhances low contrast images. Here, Contrast is the intensity value difference between the pixels. This operation can be performed in three ways; Multiplying each input pixel intensity by a constant scalar, using histogram equivalent or applying the transform which makes the dark area more darker and the bright area more brighter.

Bit-Plane Slicing: In Bit-Plane Slicing the image is considered to be as the stack of the multiple binary images and really useful in Steganography to hide the secret data. The images close to the bottom are considered to be insignificant and the images close to the top are considered to be significant.

2.1.1.2 Frequency Domain Image Enhancement Techniques

Frequency Domain Image Enhancement Techniques are based on orthogonal transform or on the convolution theorem. Here, it is supposed that g(x,y) is an image formed by the convolution of an input image f(x,y) and a linear position invariant operator h(x,y). Therefore,

$$g(x, y) = h(x, y) \star f(x, y) \tag{6}$$

Now, applying the convolution theorem,

$$G(u, v) = H(u, v) \cdot F(u, v) \tag{7}$$

Where, G, H and F are Fourier Transformation of g, h and f respectively. The orthogonal transformation has two components; magnitude and phase. Magnitude consists of frequency contents of the image while phase is used to restore the image back to spatial domain.
In frequency domain image enhancement techniques, an image is first transferred into frequency domain by Fourier transformation and various image enhancement operations are performed and then the resultant image is achieved by applying the inverse Fourier Transformation.

2.2 Image Enhancement Operations

As discussed above, Image enhancement methods or algorithms generally are either spatial domain or frequency domain and every step has its own importance according to its characteristics and its working. Some algorithms consist only spatial domain image enhancement methods and some algorithms consist only frequency domain methods to enhance the input image by means of contrast, dynamic range, noise, details, etc. Here, in this thesis a unique image enhancement method is used to enhance the image and it consists of both, spatial and frequency domain methods. Some image enhancement operations or functions used in this algorithm are as below

in detail:

2.2.1 Sobel Operator

A Sobel operator is generally used for the edge detection in the digital image processing applications. Edge detection plays a vital role in information extraction as well as for the visual perception. The better the edges of the subject and objects are, the easier its perception gets.

Now, when we talk about edge detection by using a Sobel operator, it gives an approximation to a derivative of the image. It basically separates the edges vertically and horizontally, in other words in x-direction and in y-direction. For this task a kernel or mask is used which is 3 × 3 metrics. There are two different kernels for each directions.

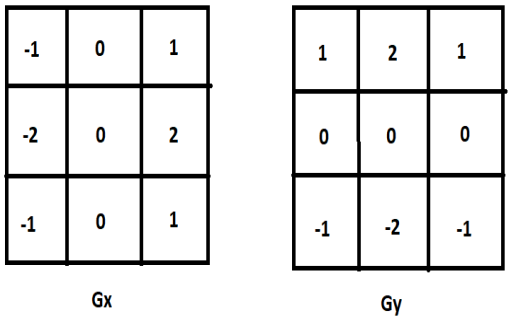

Fig. 2.2.1: Sobel Masks

Here, in figure 2.2.1 two masks of a Sobel operator are shown. Gx is a gradient or mask in X-direction and similarly, Gy is a gradient or a mask in Y-direction. In this technique, basically the mask or the gradient is placed on the image and convoluted on all over the image in order to find out the edges. Edges are basically the intensity difference of the neighbouring pixels. These both kernels can be applied to a single image simultaneously to find two separate orientations and the vertical and horizontal edges can be enhanced by finding their magnitude. An approximate magnitude is shown below in equation.

$$|G| = |Gx| + |Gy| \tag{8}$$

2.2.2 Laplacian Of An Image

Generally, a Laplacian operator is used to highlight the details of an image in image processing applications. It can also be used before sharpening operation since it highlights the details and then those details can be sharpened to make it more visible. A Laplacian operator is basically a 2-D isotropic measure of second order derivative of an image. It also directly deals with the pixel values so we can say that the Laplacian operator is also a spatial domain image enhancement method. The main difference between a Laplacian and other spatial edge detection masks like Sobel and Prewitt is Laplacian is second order mask unlike those other masks. Moreover, a Laplacian operator does not differentiate the edges of image in any particular directions like the other masks, it differentiate the edges inwards or outwards.

Laplacian Operator can be classified into two categories, Positive Laplacian Operator and Negavtive Laplacian Operator. In postive Laplacian operator the center element is negative and the elements of the boundary are positive. Similarly, in negative Laplacian operator, center element is positive and the boundary elements are negative. Positive Laplacian Operator enhances the outward edges and Negative Laplacian Operator enhances the inward edges. These two different operators are shown below in image.

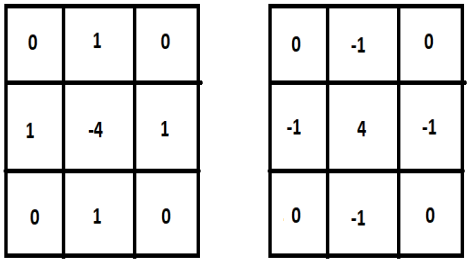

Fig. 2.2.2: Positive and Negative Laplacian Operators

We only can use one of these operators on an image at a time. Laplacian operator basically uses the gray level discontinuities to enhance the edges of gray-scale images. The Laplacian of an image having pixel intensity I(x,y) can be calculated as,

$$L(x,y) = \frac{\nabla^2 I}{\nabla x^2} + \frac{\nabla^2 I}{\nabla y^2} \tag{9}$$

A convolution filter is used to calculate this equation. These kernels are very sensitive to the noise since they are a approximation of the second derivative of the image. Thus, before using the Laplacian, the Gaussian smoothing is used to smooth the image. In other words, to reduce the high frequency noise components.

2.2.3 Addition

In digital image processing, Addition of the image is used as sharpening operator or for the noise reduction. As the images are considered to be the set of the pixel, we can sharpen an image by increasing the values of each pixels. This increment in the value can be done by adding a constant to the image or by adding the image to the image. Thus, it also makes Addition, a spatial domain image enhancement technique. An Addition operation of two images I1 and I2 can be calculated by,

$$I(x,y) = I_1(x,y) + I_2(x,y) \tag{10}$$

Similarly, an addition of a constant to an image is done by,

$$I(x, y) = I_1(x, y) + C \tag{11}$$

2.2.4 Wiener Filter

Wiener Filtering is considered to be a most important technique of a blur removal. A blur in images can be the product of the linear motion or an unfocused optics. From the image processing point of view, the linear motion is nothing but the poor sampling. If the shutter speed of a camera is too slow to realize the motion of the camera, it causes a blur in the image. Each pixel in the image represents the intensity of a fixed particular point of an image, but if the shutter speed is too slow to experience the motion of the camera, the focus gets mixed with the intensities of the other points. Wiener filtering is a frequency domain image enhancement technique as it includes the Fourier transformation of an image.

Wiener filtering approach is based on the stochastic framework in which an image is first considered to be distorted by the gaussian noise and then that image is recovered by the inverse filtering by trading off between additive noise and inverse filtering since an inverse filtering is very sensitive to additive noise.

The difference between an uncorrupted image f(x,y) and the estimated image f'(x,y) gives us the mean square error.

$$e^2 = E[f(x, y) - f'(x, y)]^2 \tag{12}$$

The approximate image estimation satisfies the minimum error function in the frequency domain. It can be shown by,

$$F'(u, v) = [\frac{1}{H(u, v)} \cdot \frac{|H(u, v)|^2}{|H(u, v)|^2 + K}] \cdot G(u, v) \tag{13}$$

The mean square error(MSE) and signal to noise ratio(SNR) in their statistical form can be given by,

$$MSE = \frac{1}{MN} \cdot \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f(x,y) - f'(x,y)]^2 \tag{14}$$

$$SNR = \frac{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} |F(u,v)|^2}{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} |N(u,v)|^2} \tag{15}$$

2.2.5 Multiplication

Like Addition, Multiplication in image processing comes with two main forms, multiplying by an image and multiplying by a constant. Generally, Multiplication is used for creating masks of an image in image processing. It is a spatial domain image enhancement technique.

Multiplication of two images can be shown as,

$$I(x,y) = I_1(x,y) \times I_2(x,y) \tag{16}$$

Multiplication of an image with a constant is given by,

$$I(x,y) = I_1(x,y) \times C \tag{17}$$

2.2.6 Gamma Correction

It is also known as Gamma Transformation or Power-law Transformation. The general form is given by,

$$S = C \cdot r^{\gamma} \tag{18}$$

S is an output pixel value and r is an input pixel value. C and γ are positive constants. A human eye perceives the light by following a approximate transfer function according to Stevens' power law for perception of brightness. According to that function, we can observe that the human eye is more sensitive to the changes in the dark than the changes in the light. But unlike the functioning of the human eye, the camera shares a linear relationship. That means if we increase the light falling on the camera by the

factor 2, the output also will increase by the factor 2. So the actual problem comes in the picture at the time of displaying the output because various displays also have their own intensity to voltage response curves which is in power function instead of linear. So, any input signal from a camera is transformed by that gamma and the result appears darker than it is expected. So, we apply the gamma correction to this signal by applying complementary voltage which cancels out the gamma effect.
To summing up the whole procedure looks like the figure shown below.

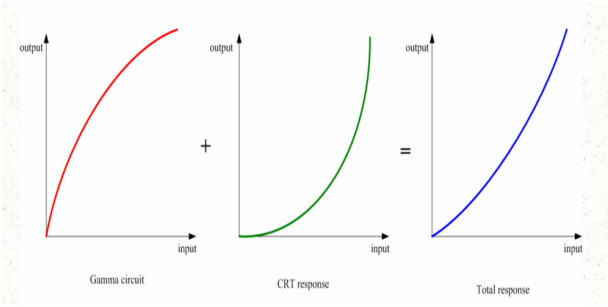

Fig. 2.2.3: Gamma correction in images[29]

2.3 Field Programmable Gate Arrays

Field Programmable Gate Arrays are widely known as FPGAs which are basically semiconductor based devices. They are mainly the matrix of configurable logic blocks and they are connected via interconnects between them. These configurable logic gates are nothing but the arrays of the logic gates, memory or other elements. Unlike the processors and ASICs (Application Specific Integrated Circuits) the user can change the functionality of FPGAs anytime by programming the hardware. This is the main advantage of FPGAs which is also known as Reconfigurability of FPGAs. These FPGAs can either be a set of simple function consisting of few logic gates or a very complex function consisting of thousands of logic gates and other elements and can acts as a comprehensive multi-core processor. This shows the capability of FPGAs which is very massive. They can be used to run millions of different operations in parallel which can be really useful in making a parallel hardware to achieve a very high speed configuration.

FPGAs offer a wide range of applications which are from video and image processing to medical and space applications. Particularly they are used as a building block for prototyping the ASICs or processors. They could be reprogrammed until the final bug-free design of ASICs before the actual manufacturing process. Even the giant companies like Intel and Nvidia use FPGAs to prototype new circuits.

Talking about the versatility offered by FPGAs, they are gaining elevation in the field of DNNs(Deep Neural Networks) which are used in the AI(Artificial Intelligence) applications since running of DNN models require a large amount of processing power. Generally, GPUs(Graphics Processing Units) are used to accelerate the processing powers but high performance FPGAs can actually outperform those GPUs when it comes to analyzing an enormous amount of data for Machine Learning. For an actual example, developers can harness the power of FPGAs without purchasing and configuring specialized software or hardware by using Microsoft services which let the developers work with common open source tools such as TensorFlow AI development Toolbox and Microsoft Cognitive Toolkit. An FPGA based design flow is shown below in figure.

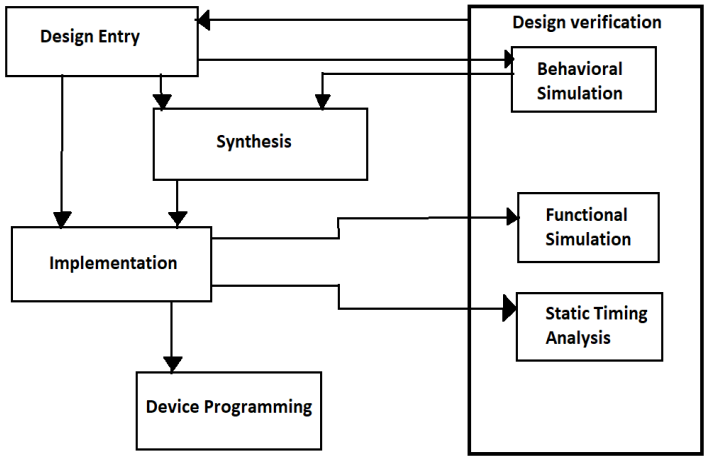

Fig. 2.3.1: FPGA Design Flow

2.3.1 Design Entry

There are multiple ways to design the FPGAs such as schematic based method, Hardware Description Language based method and the combined method which consists of both. The choice of these methods depends on the developers. If the developer seeks in dealing with hardware, schematic based method is the right choice. In dealing with the complex design or the design which can be though of as an algorithm, HDL method is better which is faster but lags in performance and density. HDL lets developers to isolate themselves from the details of hardware description whereas, schematic based method provides a wide visibility to the developers into the hardware. If one can think of a hardware design in the series of different states, state machine based method can come in handy but the tools for state machine method are limited. Each of these methods have their own pros and cons. The schematic based method is used for small designs and easy to read and comprehend whereas, HDL method is the most popular design method for FPGAs since it is faster and easy to implement.

2.3.2 Synthesis

In this step the code is actually translated into real circuit elements such as gates, flip-flops and multipliers once the design is entered in the form of code. The input in form of HDL is converted into the the netlist which consists a list of the interconnects and logic elements which are going to be used for the project. This process begins with the syntax check and then optimized by the elimination or reduction of the redundant logic once a user feeds HDL based design. This optimization leads into the reduction of the design which allows faster implementation of this design. The final step is technology mapping which consists of the connection of the design to the logic, estimation of the associated time and the production of the design netlists. FPGA synthesis is done by using the dedicated synthesis tools provided by different companies such as Xilinx, Cadence, Synopsys, etc.

2.3.3 Implementation

This step has three main steps, translation, mapping and place & route. These steps are provided by the vendors since they know better how to translate the synthesized netlist into FPGA.

The first step basically gathers all the information related to the constrains of the design that are set by the user. These constraints can be anything in terms of design, area, power consumption and time.It can be exemplify in the form of the maximum delay or the assignment of the pin positions. After that, the tool maps out the actual available resources on the FPGA board that is going to be used with the implementation by comparing them. Then it divides the circuits in form of logic blocks or sub blocks elements which results in the placement of the design into specific logic blocks which is mapped out in FPGA. The final step is to connect and route all the signals according to the set constraints.

2.3.4 Device Programming

This is the last step of the FPGA design flow. It consists of loading the routed FPGA design into FPGA by generating the Bitstream file.

2.3.5 Design Verification

The user has an opportunity to simulate and test the design at the end of every design steps of FPGA design flow but generally it is done at three essential design steps; at design entry, at post synthesis and post implementation. It would not be exaggeration to say that verifying the implemented designed functionality is very important part of the FPGA design flow.

2.3.5.1 Behavioral Simulation

It is also known as RTL (Register Transfer Level) simulation and it is done before synthesis or at design entry. This is a fast simulation which is done quickly in order

to check the functionality of the design without considering the constraints. A user can use this simulation frequently to test the code and to find the logical errors.

2.3.5.2 Functional Simulation

This simulation is performed after the completion of the synthesis. It mainly verifies the functionality of the design by ignoring the timing constraints. It is also known as netlist level simulation.

2.3.5.3 Static Timing Analysis

It is generally done at implementation and also known as timing simulation. It provides the most accurate picture of the design behavior to the user. It takes almost every constraints and the components into consideration such as wiring, delays and logic block functionality if the target FPGA. It takes a longer time since it provides the detailed simulation unlike the other steps.

2.4 HLS and Design Flow

HLS which stands for High Level Synthesis is generally known as C synthesis, behavioral synthesis, Electronic System Level synthesis or algorithm synthesis. It is a design process that creates the digital hardware implementing the desired behavior of an algorithm. It starts with the high- level specification which accepts an input as the subsets of C, C++, MATLAB, System C, etc. Then that input code is analyzed, architecturally constrained and compiled into RTL (Register-Transfer Level) design in HDL (Hardware Description Language).

The designers can build and verify the hardware design efficiently since HLS provides a better control over optimizing the design and the designer can design at every abstraction levels including gate-level, RTL-level and algorithmic-level. HLS can be performed on different platforms by using various tools provided by the different companies. Some wellknown HLS tools are Quartus II by Altera, SpeedSim by Cadence, VCS by Synopsys, Vivado simulator by Xilinx, etc.

Moreover, there are some tools that allow rapid prototyping in which users has to implement their algorithm in high level language and port those algorithms into HDL codes for the FPGA systems. For example, MATLAB HDL coder allows users to input the algorithm in form of matlab files and it creates HDL code. Also PYNQ which allows users to implement their algorithms in Python and implement them of hardware. Although these design tools use hardware libraries and overlays for this and some tools do not support every kinds of data structures. In figure, HLS design flow by xilinx is shown below.

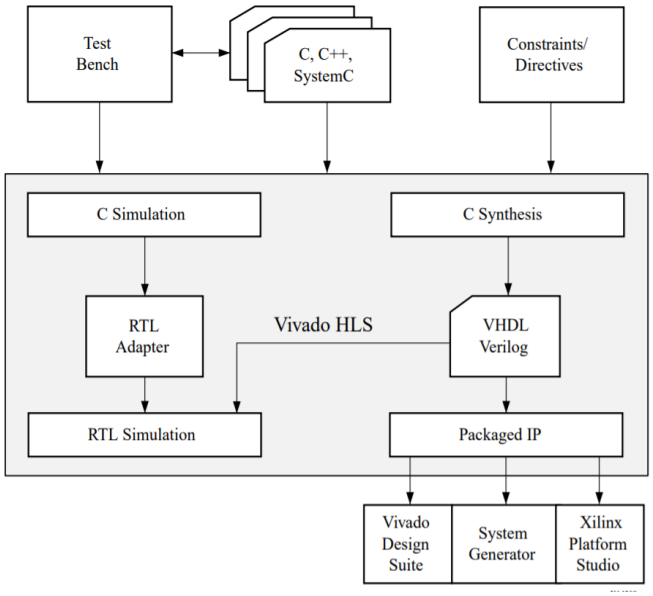

Fig. 2.4.1: Vivado HLS Design Flow[15]

CHAPTER 3

Related Works

3.1 Implementing Image Enhancement algorithms on FPGAs

The main reasons for implementing Image Enhancement algorithms on FPGAs are the ability of FPGAs to be standalone quantity to run the algorithms by providing compelling performance in terms of trading off with constraints and real time realization of the algorithms. It is desirable to have real time processing ability in any hardware which acts as a standalone quantity and performs task these days. When it comes to image processing applications, serial processors have some limitations as these applications consist of complex mathematical operations, high quality images and larger datasets. FPGAs do not only provide alternatives to implement these image processing algorithms, they also allow users to be able to trade off with hardware constraints by giving options for hardware implementation in different configurations by achieving parallelism.

3.1.1 Exploiting Parallelism

Parallelism in image processing algorithms exists in two major forms[10]; Spatial and Temporal. *Spatial Parallelism* means to divide an image in multiple parts or the sections and to process the different parts of the image simultaneously. Whereas, *Temporal Parallelism* means to represent an algorithm as a time sequence of a different simultaneous steps or operations. FPGA implementations can possibly achieve these

both types of parallelism by using the amalgamation of two methods. To exemplify that, FPGA can be configured to divide an image and separate those sections via pipelines and all of those sections can be processed simultaneously.

Although the degree of parallelism depends on the processing mode and the hardware constraints introduced by the system. These constraints can be classified in three parts; Timing constraints, Resource constraints and Bandwidth constraints. These constrains make the mapping of the algorithm to desired hardware more difficult as they are inseparably linked with both the approaches that are low-level approach and high-level approach. The mapping of an algorithm to hardware is basically a technique to specify any image processing algorithm in any hardware language that can be compiled into a netlist.

Talking about the parallel hardware, in [32] the authors have successfully designed a highly parallel architecture model of the algorithm for enhancing the retinal fundus image to achieve high throughput. Their algorithm is based on brightness control, contrast stretching and histogram equalization as well as they have achieved temporal parallelism. For example, the brightness control algorithm has been implemented in parallel to maintain the same brightness level in image for multiple pixels. This algorithm is designed by interconnecting the different blocks such as constant, compare to the constant, add, etc. The constant block generates different constants which are compared by the different inputs by the compare to constant blocks. If the comparison is true the output is 1 and if the comparison is false the output is 0. The view of the input values is enabled by the display block which accepts real and complex signals. The input value is taken as the pixel value as the constant, it is added to the brightness factor and compared with 255. If the addition is less than 255 than it is taken as the output otherwise it is adjusted to the maximum value 255.

The algorithm is mapped using high-level mapping approach and MATLAB Simulink platform is used for interactive graphical environment since it provides the set of block libraries for design, simulation and implementation. Then the Bit-file is generated for the hardware implementation. The hardware implementation is done on Spartan 3E FPGA of Xilinx family and for software development Xilix ISE 14.1 tool is used.

3.1.2 Low-Level Approach

At this level of implementation, a designer has to deal with all the constraints manually and to achieve this level of abstraction the design must be specified at the RTL (register Transferred Level) to maintain the better control and optimization. This approach provides a better control over the constrains and flexible optimization since it deals with the low-level languages such as VHDL and Verilog. Success and significant speed increment over a sequential architectures have also been shown in the low-level mapping which is used in this approach[13].

The better control feature of this approach refers that the user has to be ready to deal with the hardware constraints and its implementations directly. The timing constraints of real time image processing applications introduce some additional complexities such as memory bandwidth, resource conflicts and the necessity for the pipeline.

Although, design at this level of abstraction requires an intimate hardware knowledge since programming for RTL for hardware at this level is similar to assembly level language in software domain which is considered to be difficult and time consuming due to the usage of complex image processing algorithms and larger image sizes. So, this approach is used to implement very small applications. Moreover, these days there are number of tools available in market which make easy to map the algorithm to hardware in very little time and they also allow the users to program in high-level languages which basically hide the low-level details. Because of this this approach is not widely used.

Talking about the low-level mapping[30] the authors have successfully performed the hardware implementation of a simple yet effective image enhancement method for automatic vehicle number plate detection. Their method was based on sharpness control, brightness control, histogram equalization and contrast stretching. The main motive of this method of image enhancement method was to improve the image quality which is going to be used in some specific application like Optical Character Recognition(OCR). The main cause for poor image quality is low light in this application and

an alternative option which is to use a flash light with camera will raise problems for the drivers. The algorithm have been implemented in form of different blocks such as sharpness control, brightness control, contrast stretching and histogram equalization. These blocks have been implemented by using Flip-Flops, multiplexers, comparator, adder, multiplier, etc. Xilinx synthesizer for FPGAs by using verilog HDL codes has been used to generate the hardware architecture according to the arrangement. They also have provided separate block diagram for each operations referring their internal architectures. They used Verilog HDL for all the coding and synthesis that on Xilinx ISE v14.7. A Xilix Virtex V has been used to verify the hardware implementation. Power calculations were done by using Xilinx power estimator tool. Along with this timing analysis and device utilization report have been shown. The only limitation of their method was limited image size 500x200 pixels due to limited memory.

3.1.3 High Level Approach

Unlike the low-level approach, the high-level approach is associated with the high-level languages and their compilers based on their popular software languages[10]. The ultimate motive of these high-level languages is to hide low level details from the developers such as pipelines and scheduling. In this approach the parallelism and scheduling is exploited by the compiler based on various optimization techniques such as loop unrolling to exploit spatial parallelism and automatic pipeline to exploit temporal parallelism. So, for hardware designing, this high-level approach would be easy and simple for the users to design any hardware in short time of period without engaging in the low-level details since the users only have to make few changes in the existing programs, compile it and to generate a resulting FPGA file. This allows a more algorithmic approach to a hardware design and appears to be a perfect solution for image processing, which already has a large stable code base of well defined software algorithms for implementing so many common image processing operations[10]. It also makes mapping of netlist from algorithm easier for the users who do not have that much knowledge of low-level issues of hardware design. Although, this hardware design by using software turns out be unwise in some cases when it comes to imple-

menting some mathematical expressions such as multiplication and square root since they turn out to be very costly in hardware design. From software point of view they are too easy to exploit because it reinforce the software mindset but they are costly at the hardware end. So, in some cases it may not represent the best algorithm to used for some specific processing modes of FPGAs and this algorithm may need to be rewritten to meet the hardware constraints which leads to low-level mapping.

Authors in [24] have used the high-level approach to map the multi-directional Sobel algorithm to FPGA. According to their paper, hardware implementation of edge detection algorithms for real time image processing applications is one of the major issue in the field of hardware implementation as edge detection technique plays an important role in texture analysis, image segmentation and pattern recognition. In image processing applications generally Sobel operator is used for edge detection. So the authors tried to propose an FPGA based hardware architecture of real time edge detection technique based on Sobel operator. In their proposed architecture the edges of an image can be detected into four different directions; vertical, horizontal, left diagonal and right diagonal. Moreover, there are two outputs of the algorithm that are resultant of vertical and horizontal directions and resultant of left diagonal and right diagonal direction. These output are used according to the necessity of the applications. The size of input image is taken as 1024 x 1024 pixels in gray scale. The input to their algorithm comes from a camera in form of serial single bits of pixel by a synchronous clock and entered into 3 x 3 arrays. Then gradient calculation is performed by three cascade shift registers. After that in the window processor block four different gradients are calculated simultaneously and the output of this block is given to the comparator block in the form of positive, negative or zero. Now, the comparator block has main three functions; taking the absolute values of the four inputs, calculating the resultant values and to perform the thresholding operations on all results so that they do not exceed 255. The data flow is controlled by the the controller block which also decides when to save the data. Apart from that it also controls the counter which generates addresses for the RAM block.

For simulation they have used Xilinx Spartan3 XC3S200 FPGA chip. For synthesiz-

ing the VHDL modeling codes the ISE9.2 and MATLAB have been used and then device utilization summary has been shown. This high-level approach was successfully employed in their method since it was very simple and small algorithm for edge detection technique. Moreover, the power consumption and timing constraints were not considered in their method.

3.1.4 Design Patterns: A more flexible solution

The both mapping approaches discussed above have their own disadvantages. High-level approach reinforces software mindset which results in less optimal mappings. Whereas, low-level approach helps in overcoming the shortcomings imposed by high-level approach but is it more labour intensive and puts more emphasis on the hardware reusability which has lead to some common challenges. Design patterns proposed in [13] is a common methodology for design in software engineering but it taken to the image processing on FPGA from architectural engineering. Design patterns basically are the pattern which correlate the possible mapping techniques used to manage the constraints as we envision applying them. Then it focuses on the key elements of the solutions that may possibly be reused in mapping. A way to convey the design experience is provided by the recorded patterns in informative and structured manner by locking the essence of solution. These generalized solutions offer suitable abstraction for hardware and emerging languages in this fast paced area because their usage is not restricted to any particular suitable platform or language. In this approach while dealing with some complex mathematical operations such as square root and multiplication which can possibly be expensive and lead to an inefficient usage of resources, generally precalculated expression values are used. Those values are generally stored in look-up tables(LUTs) in this method which is general enough so that we can use it for any future situations which uses complex expression multiple times. Although, this design patterns are more useful to the problem which is occurring over and over. So it is more suitable for the instance when the mass production is being done of the hardware.

3.2 Fixed Floating Point Configurations and Their Trade-off

Architecture wise, the digital signal processing units can be commonly classified in two parts; floating point configuration and fixed point configuration. These configurations refer to the format which is used to store and manipulates the numbers or the pixel values in the device. Now that most of the image processing applications consist of complex algorithms which are generally based on the mathematical operators, the designers ought to consider the ways for storing the values and manipulating them. From these two types fixed point configuration allows higher precision as per the bit-width representation [23] but not all the cases are about only higher precision, sometimes data representation also plays an important role especially in the high performance computer system to achieve suitable dynamic range. Thus, in this case floating point configuration provides a feasible solution. Although these floating point algorithms are generally implemented on software and run on microprocessors, which results in performance penalty in data transfer between ALU and program memory Talking about image processing applications using FPGA, also known as *Bottleneck Problem*. This problem can be overcome by the usage of multi-core processor. Nowadays, GPUs are the microprocessor based solution and they improve the performance by working with higher frequency, though results in a bottleneck problem and higher power consumption which is a major drawback. Now, talking about the implementation of the complex algorithms on FPGA, the designers can improve the performance by using the hardware resources and capabilities of parallelism of FPGA. But, ideally only fixed point arithmetic is provided by FPGAs which in turn binds in limited dynamic range. Thus it is up to designers to trade-off between the cost in area and the bit-width representation. But nowadays, many platforms such as Intel, Xilinx and MATLAB provide users to efficiently use the floating point configuration for hardware implementation for impactful DSP designs considering FPGA as suitable environment to implement floating point algorithms due to parallelism and pipeline.

In [4] authors have compared fixed and floating point configuration for the implementation of their deflectometry algorithm and acclaimed that the fixed point configuration is efficient for FPGAs. Mainly, in deflectometry an automatic analysis of the reflected patterns of the target surface is performed which can be useful in such applications as to finding the irregularities in surface area and collecting the topographical information of target material. Their algorithm basically analyses the series of reflected image patterns and determines the defects or irregularities on the surface. These patterns are generated by using camera and reflected on the computer screen in form of horizontal and vertical fringes. Zynq based internal architecture consisting all blocks such as memory, controller, and processing has been proposed along with the FPGA based architecture of their method. They used Xilinx HLS design tool for rapid prototyping and synthesis. After that, Vivado design suite was used for the complete design and Xilinx SDK was used for software design.

Mainly two approaches were used in their design; 32-bit fixed and floating point as well as the timing and utilization for each approaches were compared and shown. They have acclaimed that the fixed point configurations were 35% faster in terms of execution time than floating point approach. Moreover, the fixed point configuration in their method consumed lesser, almost half of the resources as compared to the floating point configuration.

CHAPTER 4

Methodology

To tackle the problem statements proposed in the first chapter we would like to propose these following methods. First, to overcome the limitation of the image enhancement algorithm due to the usage of the Laplacian Operator, we would like to use Local Laplacian operator in order to make the image enhancement algorithm useful for even coloured images. Moreover, it would help us in improving execution time as the Fast Local Laplacian operator is faster than the Laplacian operator in terms of run time. Secondly, to overcome the performance problem of Sobel operator in edge detection application we would use LoG operator which would improve the effectiveness of edge detection. At last as resource management, power consumption and execution time are very important parameters in hardware implementation, we would like to implement an image enhancement algorithm on parallel hardware to achieve better power consumption and execution time. For the initial implementation, Python3 platform has been used to verifying different image enhancement operations of the algorithm. For RTL simulation MATLAB has been used as it is one of the best platform to implement image processing algorithms because it offers an ease in processing 2D and 3D data or image and provides different DSP (Digital Signal Processing) blocks to develop algorithms and design. To test, verify the generated HDL script generated by MATLAB Coder on a targeted hardware, Quartus Prime Lite has been used. Quartus Prime provides optimization and physical synthesis which makes the design performance better in terms of area and compilation time. The implementation flow of this algorithm has been shown below in the figure.

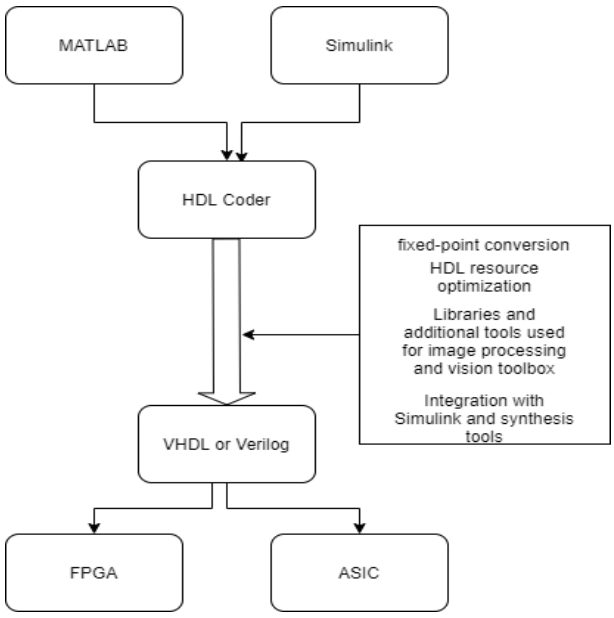

Fig. 4.0.1: Hardware Implementation flow using MATLAB

4.1 Overcoming the Limitation of Laplacian Operator by Edge-Aware Image Processing

Edge detection techniques are indisputably important in image processing since they link with different applications with different domains. Generally, in image enhancement methods Laplacian operator is used for edge detection or in simple words making the edges more appealing in the beginning of the algorithm. Laplacian Operator is basically second order derivative of an image which can be developed by convolving the mask around the targeted image whose kernel computes the addition of the weighted gray level differences between the pixel and its neighbours. So, its performance in image enhancement of gray scale images is remarkably good but its application is only limited to gray scale images. This operator can be extended by applying it to each separate R, G, B component and combining those results to bring in the sharpened coloured image. But, it has been reported that the simple extension like this is not the best possible solution[16]. So, to overcome the mentioned problem, Local Laplacian operator really comes handy as it operates on the principle of edge-aware

image processing.

The Edge-aware image processing aims to modify the input image I into output image I' in order to maintain or retain the large discontinuities of image I i.e, edges. Moreover, it also aims to retain the overall shape or profile of those discontinuities without making edge transitioning smoother or sharpen irrespective of the increment or decrement in the amplitude of the edges. This edge-aware image processing technique is significant for some techniques that deals with manipulating the images in spatial domain. Although if the user fails in implementing these techniques, the result would cause haloing, shifted edges reversals of gradient, etc. The local laplacian operator has been developed based on the pyramid-based edge aware filtering to provide a further approach to edge-aware image processing, using a multi-scale representation using Gaussian/laplacian pyramid or a wavelet decomposition[17]. The author of [17] has proposed an algorithm to implement the local laplacian operator based on the Laplacian pyramid.

4.1.1 Local Laplacian Filtering Algorithm

Giving an overview of the algorithm, an image is fed as an input and then each pixels of Gaussian pyramid of the input image is mapped by pointwise function and then intermediate Laplacian pyramid is generated from these intermediate results. At last the appropriate pixel is copied in the output Laplacian pyramid. The whole process is repeated over all scales for every pixel of the input image until the pyramid is filled which then gives the final result. The user can generate only some intermediate parts of the image for more efficient computation instead of the whole image. The general flow of the Local Laplacian algorithm has been shown below in the figure.

The major available components of this algorithm are α, β and σ. A user can freely manipulate the details of the targeted image without specifying the decomposition of an image into these components proposed by[28]. These components have their own functions and are used to control the output such as:

- α controls the details of the image. if $\alpha > 1$ then it depicts detailed enhancement

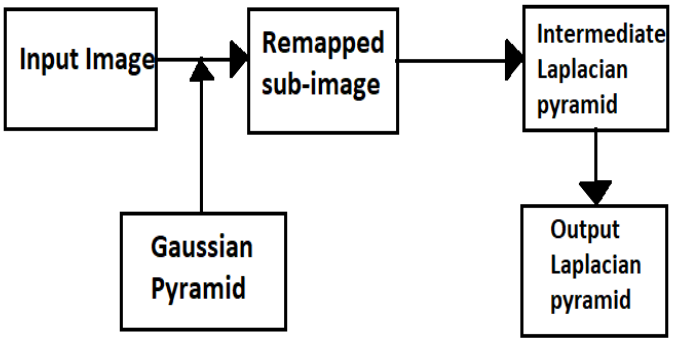

Fig. 4.1.1: General Flow of Local Laplacian Algorithm

and $\alpha < 1$ then it depicts the detailed smoothening.

- β controls overall appearance of an image. $\beta > 1$ depicts intensity range expansion and $\beta < 1$ depicts intensity range compression.
- σ is a constant which gives a threshold to fix what comprises the details.

First of all Gaussian pyramid G for each levels$(l_1, l_2, l_3..)$ of the input image I is created from bottom to top, i.e., $G_0, G_1, ..., G_l - 1$. Then the Laplacian pyramid L is computed from top to bottom, i.e., $L_l - 1, L_l - 2, ..., L_0$. The output O of this algorithm is the collapse of all the Laplacian pyramid levels. The function is shown as below.

$$Q_k, q = LocalFilter_{\alpha,\beta,\sigma}(I_k, q) \tag{1}$$

The algorithm to implement local Laplacian proposed by [17] is shown below.

Algorithm 1 Local Laplacian Filtering

Input : Image I, α, β, σ
Output: Image O
Construct the Gaussian pyramid of I (with l levels)
for level G_l in pyramid **do**
  Proceed LocalFiltering and apply remapping function
  Compute output Laplacian Pyramid L_l
end for
Output pyramid $O \leftarrow$ collapse (L_l)

Fig. 4.1.2: Pseudo code of Local Laplacian Filtering[28]

4.1.2 Sampling for acceleration

As the general idea of this algorithm is to propagate the edge information of the neighbouring pixels, we can easily do that if the pixels are near to each other but if the pixels are far it will consume too much time to make the uniform grid calculations. So, the author of [28] proposed a pattern in which the red original pixels can choose those pixels to propagate their edge information instead of propagating for every pixels in the neighbourhood of an image. For example, if the offset of pixels is (δ_x, δ_y) we only need to calculate the local filtering for the points that hold the condition shown below.

$$\delta X^2 + \delta y^2 = 2^k, k \in N. \tag{2}$$

Now, it is easy to find that number of points we are going to sample since it obeys O($log^2 M$) from an $M \times M$ sub-window. Now, the time complexity is improved from O(N log N) to O(N log log N).

4.2 LoG to Solve the Performance Problem in Edge Detection

Edge detection is very important in image segmentation. It is much more than isolated points and lines since it contains the basic characteristics of the image and it pacifies the data amount as well as it filters out the useless data. In image processing domain an edge is basically a set of connected pixels which separates black and white regions. This process of detecting the edge is generally carried out by the first order derivative operator. Its common applications are to find the surface discontinuities, highlights, colour or texture, etc.

Laplacian or Gaussian operators are used in image processing for smoothening the image but, generally Sobel operator is used for edge detection in image processing applications as it is very simple and effective. It is used to detect the edges along with logical OR and AND. It used 3 x 3 mask on the image and often used as a detector of the vertical and horizontal edges. Though, it offers poor Signal to Noise Ratio(SNR) because with the increment in noise, the gradient magnitude of the edges decreases which produces poor results also it is ineffective for the detection of the edges in other directions than horizontal and vertical. So, to overcome this performance problem in edge detection technique we would like to propose an alternative method by using Laplacian of Gaussian(LoG) operator which is more efficient and effective in edge detection in image processing applications.

Unlike the Sobel operator, LoG operator offers the usage of 3 x 3 and 5 x 5 masks to the users. It is a combination of Laplacian and Gaussian noise. LoG has the same property as a linear differential operator but it works in every direction. It is a second order derivative which only finds the edge magnitude. The implementation technique of LoG operator is very straightforward. First, an input image is smoothened by applying Gaussian and then it is filtered by the Laplacian filter. Laplacian filter is the second order derivative. It is very sensitive to the noise so that we could say that it is really effective. It works remarkably good in some image processing applications in which the targeted image contains very low pixel values. The steps to implement

a LoG operators are shown below.

- Convolve the LoG kernel directly to the targeted image or first apply or first apply Gaussian and then apply Laplacian
- Find the zero crossings in the image
- Set an appropriate threshold to extract only strong edges of the image.

4.3 An Improved Image Enhancement Algorithm

To achieve an efficient and effective image enhancement operation on the targeted images we would like to propose an algorithm which basically shows an improvement over the algorithm proposed in [21] which has been improved by overcoming the performance problem of Sobel operator and overcoming the limitation of the Laplacian operator. It was only designed for the enhancement of the grey scale images since Laplacian operator was used and now it has been replaced by a fast local Laplacian operator which helped in extending the area of application of this algorithm. Moreover, the performance problem in edge detection also has been solved by using LoG operator. Thus, this improved image enhancement algorithm can be used to serve the main purpose of improving human-computer interaction and used in various domain of image processing such as computer vision, feature extraction, medical image analysis, etc. The general diagram of the proposed algorithm is shown in figure below.

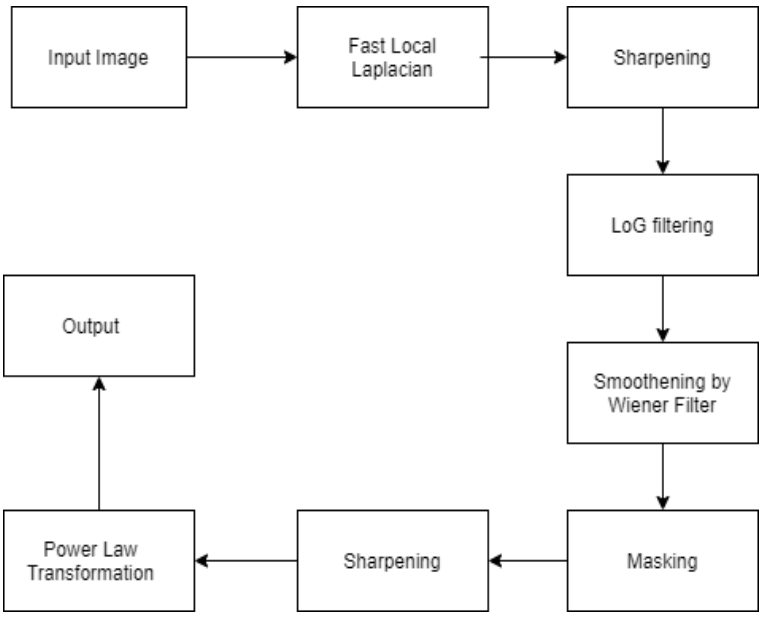

Fig. 4.3.1: An Improved Image Enhancement Algorithm

The input of this algorithm is ideally considered to be an image with standard size 1920 x 1080p. It is assumed that the image has come into contact of noise and it is to be displayed or used in certain applications of image processing such as medical image analysis, computer vision, feature extraction, etc. First of all, the details of degraded input image are highlighted by using Fast Local Laplacian filter which works on the principle of self aware image processing. So in this initial step the intensity of the details are increased or decreased according to the necessity without changing the overall appearance of the shapes in the image. Moreover, it is fast since it uses the sampling technique when it comes to propagating the pixels detail. After that, in the second step the input image is sharpened by the addition of the filtered image acquired after the first step and the input image itself. This sharpened image is then provided to the LoG filter to enhance its edges where the prominent edges of the sharpened image are enhanced in way that the output images of step three are much more dominant than the output of the step first. Next, in the step four Wiener filtering is used to obtain smoothened gradient image which appears to be much brighter than the image at the previous step. Now, a mask is generated by taking the product of the smoothened gradient image and the sharpened image in step five to provide a

user more enhanced image analysis. This mask and the sharpened image resulted from step two are added to get the final enhanced image which turns out to be more sharpen than the images at the other steps. At last, Power law transformation is applied to the image acquired from the final step to increase its dynamic range so that it can be used in any image processing applications of directly displayed on the screen.

When it comes to implementing this algorithm on a hardware by using MATLAB, MATLAB eases the task of a designer by providing some readymade building blocks for certain image processing operations. In this these thesis those building blocks of MATLAB provided by Computer Vision Toolbox and Vision HDL toolbox. For example, considering image sharpening step of the algorithm a designer can also add noise removal block in image sharpening block and save time and device resources by eliminating the subsequent image smoothening step from the algorithm. Similarly, in this proposed algorithm Wiener filter is used for noise removal and image smoothening but since while implementing the image sharpening operation noise removal block is added to the design, the redundant step of Wiener filtering has been discarded.

4.4 Introducing Parallelism

As we saw the flow of this algorithm was sequential which lacks parallelism. When it comes to implementing the algorithms on the hardware, the user ought to consider thinking about this since it directly deals with the most important parameters such as execution time and power consumption. So we would like to propose a task parallelism of the proposed algorithm. Although, this algorithm has lots of interdependent steps which affect the whole image and only makes temporal parallelism of the algorithm possible. The task parallelism for the proposed algorithm is shown below according to its dependencies.

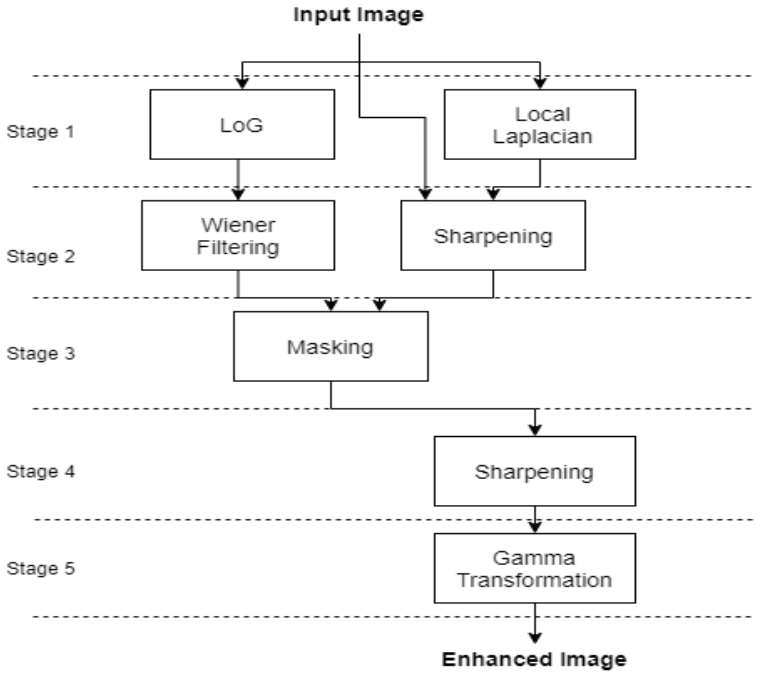

Fig. 4.4.1: Task Parallelism in proposed algorithm

In above figure different steps of the algorithm show the different steps of the algorithm have been shown according to the flow of algorithm and their dependencies. So instead of taking seven stages algorithm finishes in five stages. For this a user can either use a multi-core hardware of any other parallel hardware such as FPGA. In this research FPGA has been used to achieve task parallelism.

4.5 Implementation Strategy

For the initial implementation of the algorithm Python 3.8 platform has been used as it provides an ease to implement image processing operations by offering various DSP libraries. For the final implementation, High Level Synthesis has been used. For Register Transfer Level (RTL) simulation, MATLAB 2020 has been used. First, the algorithm was implemented on MATLAB again in order to achieve this. MATLAB offers an additional software tool named HDL Coder which let users to perform HLS and generate portable and synthesizable Verilog or VHDL code from state flowcharts, MATLAB functions and Simulink models. HDL coder uses workflow advisor which

automates the programming of Xilinx, Intel and Microsemi FPGAs [14]. Moreover, the designers can control the HDL implementations and architectures along with highlighting the critical path of the system and generating resource utilization for the targeted FPGAs. Although in this thesis, for implementing the improved algorithm on a targeted FPGA Quartus Prime Lite software has been used.

The steps for implementing any image processing algorithm on FPGA by using MATLAB coder are shown below.

1. Install the MATLAB license version including its add ons for image processing toolbox and FPGA designer such as HDL Coder, Vision HDL Toolbox, Computer Vision Toolbox, Fixer Point Designer, etc.

2. Create a behavioral model of the algorithm and then create a test bench for that model. There is a specific structure for the test bench in MATLAB that includes the main function and the test file in which the function is called. This behavioral design model also can be generated in Simulink if the designer wants to deal with hardware domain.

3. After creating the behavioral model and test bench, verify them by debugging them by using MATLAB and/or Simulink.

4. Once they run error-free, open the HDL coder toolbox from the external add-ons and open those two files made in the initial step. This step will not be there if the designer has chosen the Simulink design flow since Simulink will directly open the design model in HDL coder.

5. Now by using the Work flow advisor provided by MATLAB, set the design flow of the targeted model. The Work flow advisor offers different options such as, definition of input type, fixed point conversion and optimization, selection of code generation target, HDL code generation and HDL code verification.

6. After the generation of HDL code, the HDL code generator creates the reports including the traceability, Model web-view, Resource utilization, High level tim-

ing critical path and Optimization report which allow the user to envision the results of the implementation of the algorithm on targeted platform.

7. Now, for optimization and physical synthesis use the synthesizable HDL scripts created by MATLAB Coder on Quartus Prime Lite software and generate the different reports of the synthesis.

Although, there are some MATLAB functions which can be directly used in behavioral model of the algorithm but when it comes to the HDL generation and fixed point conversion they result in error saying "The function is not supported for the HDL code generation". This can be dealt with by using manual functional block in Simulink or declaring those fucntions as extrinsic functions in MATLAB test bench. But, then those extrinsic functions will be discarded from the HDL code generation process and only executed on the MATLAB environment. Moreover, there are some ready-made building blocks provided by the Vision HDL toolbox and the Computer vision Toolbox for some image processing operations that can be manipulated in dedicated operations of a particular image processing algorithm and can be used with HDL coder.

4.5.1 Streaming Pixel Interface: for feeding an image as an input

As we all know that we can not feed an image as an input to any FPGA devices, since it does not accept the image and offers very high cost in terms of area and memory when it comes to processing an entire frame of video at a time. That is why in this thesis "Streaming Pixel Interface" offered by Vision HDL Toolbox has been used. This streaming pixel interface works on a principle of serial processing and instead of operating on a whole frame it operates on a pixel, line or neighbourhood pixel. Some blocks of MATLAB even offer multipixel streaming (with 4 or 8 pixels per cycle) for operating on high resolution videos. Although, HDL generation for multipixel streaming is not supported with system objects [16].

This video capturing system basically generates the interactive intervals between lines and frames as it scans the video signals from left to right from top to bottom. This interval is known as horizontal blanking interval and it is split into the front porch and the back porch which relates to the number of the samples between the end of the active line and the synchronize pulse as well as the number of the samples between the synchronize pulse and the start of the active line respectively. Also, this pattern needs start and end signals for each directions. Vision HDL toolbox allows a designer to configure the size of the active and inactive frames.
For implementing this interface a designer needs two blocks which are Frame To Pixels for processing and Pixels To Frame for output. MATLAB and Simulink both provide facility to use this blocks. MATLAB provides a command to configure these blocks and Simulink provides thee blocks under the Vision HDL Toolbox.

4.6 Strategies For Results, Verification And Quality Estimation

The aim of this step of research is to show the improvements of this proposed algorithm in terms of time and resource utilization. Moreover, The hardware used for the implementation of this algorithm is intel core i7-10700 CPU @ 2.90GHz. For hardware implementation the generic resources of HDL coder and Workflow Advisor have been used since targeting a specific FPGA board requires two additional support packages which are HDL Verifier and HDL Verifier Support Packages for FPGA Boards. For a targeted FPGA synthesis and implementation, Quartus prime Lite has been used. The implementation has been done on a "Cyclon V: 5CEBA2F17C6" FPGA.

4.6.1 Verification

The verification of the algorithm is done by the HDL model advisor according the HDL coding standards provided by the MATLAB and the designer can get the HDL coding standard report generated by the HDL model advisor before the generation

of the HDL code.This report can either be generated from the command line using the specific command provided by MATLAB or during the HDL code generation task with the help of the HDL Workflow Advisor.

4.6.2 Time Analysis Methods

FPGA in Loop block of HDL workflow advisor provides the options to achieve the compatibility check of the generated HDL model which also has the internal configuration to measure the execution time of FPGA without using any external hardware. The designer can set overclocking factor to 1 under the runtime options which means the input is sampled by the FPGA's internal board clock and when the output frame size is selected as inherited, the output frame is as same as the input frame.The second method is the usage of Tic-Toc. In this method a user can start the stopwatch under the Tic which also records the internal time and read the elapsed time by using the Toc function.In this method a user needs to read the time which is displayed by the CPU for the MATLAB application. Moreover, adesigner can use the "timeit" function offered by MATLAB to measure the run time of any specific functions.

4.6.3 PSNR for the quality estimation

PSNR is the most commonly used method for measuring the quality of the image compression. In this case, an image is an original data and the noise is the error introduced by the compression and when it is compared to the compression codecs, it refers to an approximation to human perception of reconstruction quality."The typical values of PSNR for 8 bit depth are between 30 to 50 dB for lossy image and video compression. For 16 bit data PSNR values are between 60 to 80 dB." [10][5] PSNR which is widely known as Peak Signal to Noise Ratio is a technical term for the ratio between the maximum power of any signal and the power of the noise which corrupts the signal. PSNR is generally expressed in decibel scale since many signals have a very wide dynamic range. It is generally defined by Mean Squared Error(MSE). PSNR for a noise free gray scale image I with m x n resolution is given

by the equation shown below.

$$MSE = \frac{1}{mn} \cdot \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} [I(i,j) - k(i,j)]^2 \tag{3}$$

$$PSNR = 20\dot{l}og_{10}(MAX_I) - 10\dot{l}og_{10}(MSE)$$

(4)

Where, MAX_I is the maximum pixel value of the image. Moreover, for the coloured images the value of PSNR is same with three RGB values per pixel except the MSE which is the sum over all squared value differences divided by the image size and three.

CHAPTER 5

Experimental Results And Analysis

Based on the methodology discussed in the previous chapters we developed the MATLAB behavioral models first for every steps of the algorithm and developed the testbenches and functions for every steps or operations of the algorithm. We also tried to build design models in Simulink for some operations which involve the functions that are not supported by the HDL coder. Then those design models were passed to the HDL coder and HDL codes have been generated for each steps. Then the algorithm was implemented on the generic hardware platform provided by MATLAB. The results have been acquired in terms of run time and PSNR. Moreover, Quartus Prime Lite has been used for optimization and hardware realization.

5.1 Implementing Local Laplacian operator

The implementation of the Local Laplacian is a bit arduous since it involves a lot of functions that are not directly supported by HDL coder for the code generation and their manual implementation on Simulink increases the complexity tremendously. We cannot even just discard the code generation for those function since they are going to be implemented on the hardware. So for the operation like this, we have used a method that involves the generation of the subsystem. A subsystem is just as same as the other MATLAB or Simulink models which resides in a model as its one of the functions and connected with the other functions. The diagram which gives the

implementation overview of the Local Laplacian is shown below.

Fig. 5.1.1: Overview of Local Laplacian

As we discussed above, three blocks(Gaussian pyramid, Laplacian pyramid and reconstruction) from above diagram has been included to the generation of the subsystem so that the main model looks uncomplicated. Moreover, since the model is going to be implemented on the hardware we ought to design a model as if it is a hardware. So for processing the image we would require the pixel streaming interface which is in the main model. Apart from that we would require something to store or display the result if we were designing a hardware according to this algorithm. So, the image has been saved to the MATLAB workspace. The main model of the is shown below in the figure.

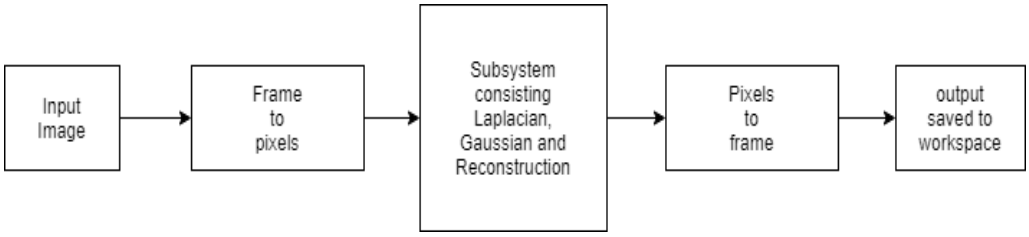

Fig. 5.1.2: Main MATLAB model of Local Laplacian

Here, for the generation of the subsystem, Computer vision Toolbox and Vision HDL toolbox are used as they provide some ready-made building blocks for some image processing operations and pixel streaming interface. First, the behavioral model of Local Laplacian was generated and verified. Next, by using the building blocks provided by the additional toolboxes the model was generated in Simulink. MATLAB provides a feature to convert a MATLAB script into a Simulink model and vice versa. So, for the subsystem the MATLAB script has been uploaded to the Simulink block in order to realize the working of the model. After the debugging, the model was finalized by solving some errors. Then by using HDL coder the HDL code generation

and hardware implementation was done as well as the code generation report was generated which is shown below.

Resources	**Numbers**
Multipliers	208
Adders/Subtractors	253
Registers	633
Total 1-bit registers	4230
RAMs	9
Multiplexers	169
I/O bits	50

Table 5.1.1: Generic Resource Utilization Report of Local Laplacian

The table shows the generic resource utilization report of the implementation discussed in this section. Since, Local Laplacian includes the generation of Laplacian and Gaussian Pyramids as well as the reconstruction of the image, the resource utilization offered by this algorithm is not so efficient. Moreover, the debugging time and development time is quite longer which are 72.01 and 97.2 seconds respectively. But the run time is 10.34 seconds which can be considered to be good.

The resource utilization for this operation on a targeted FPGA after optimization has been shown below.

Resources	Numbers
Logic Utilization (in ALMs)	2797/9430 (29.6%)
Total registers	10450
Total pins	46/128 (36%)
Total block memory bits	96051/1,802,240 (5.23%)
Total DSP blocks	2/25(8%)
Combinational ALUT usage for logic	670

Table 5.1.2: Resource Utilization Report of Local Laplacian by Quartus Prime Lite

5.2 Implementing Sharpening and noise removing operator

In the main proposed algorithm Addition and Wiener filter have been used for sharpening and noise removal respectively. But as MATLAB and Simulink provide ready-made building blocks for image processing operations by allowing designers to use the additional toolboxes, here in this thesis Noise removal and sharpening block has been used in Simulink to desing a subsystem for sharpening and noise removal. By doing this, we can eliminate the blocks of Adder and Wiener filter and as a result we can save resources of the hardware device on which this algorithm is going to be implemented. The overview of the Sharpening and noise removal model has been shown below.

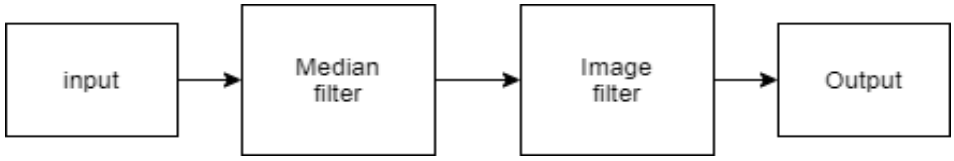

Fig. 5.2.1: Overview of Sharpening and noise removal model

The implementation of Noise removal and sharpening model takes place in two different blocks, i.e., filters which are median filter and image filter respectively. The

building blocks provided by the Computer vision toolbox have been used. First the image is read into the input block and salt and pepper noise has been added to it. Then median filter is used for the noise removal. After that, the image filter block is used for sharpening. The image filter block comes from the Vision HDL toolbox, which also can be used for different operations such as blurring, sharpening, detecting edges, etc. For configuring this building block for different operations designers just have to change the coefficients and other configurations which is one of the perks of using MATLAB for hardware implementation. This feature also shows the reconfigurability of the blocks.

Now, as we know that the Vision HDL Toolbox operates on a single pixel instead of operating on a whole frame like FPGAs, we would need to convert the frames into pixels by using the pixel streaming interface. But this pixel streaming interface is consisted in the main design model unlike the filters. So, basically the main model of this algorithm looks like the figure shown below.

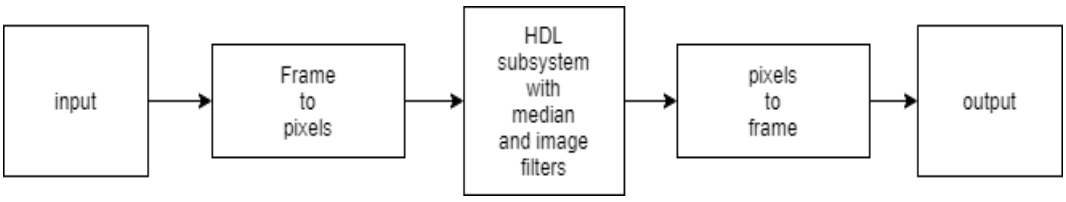

Fig. 5.2.2: Main MATLAB model of Sharpening and noise removal

This model is also was generated similarly as the local laplacian model. First, the algorithm and the testbench were tested on the MATLAB software by creating the behavioral model. Next the Simulink model and the MATLAB testbench for HDL code generation were created. After debugging the fixed point conversation was done and then the HDL code was generated by using HDL coder and finally the design was implemented on the generic FPGA offered by MATLAB.

The report generated by the code generator is shown below. Since the reconfigurable blocks provided by the Vision HDL Toolbox and Computer Vision Toolbox have been used to design the model, the resource utilization is more efficient for this model as compared to the Local Laplacian model. Moreover, this output functionblock also

offers the verification of the output in terms of PSNR. But as the Gamma Transformation is the last step of this algorithm, PSNR is calculated after that step. The simulation time and generation time for this model are 13 seconds and 28 seconds respectively which are also decent. Moreover, the run time for this implementation is 5.2 seconds.

Resources	**Numbers**
Multipliers	3
Adders/Subtractors	37
Registers	509
Total 1-bit registers	2539
RAMs	4
Multiplexers	175
I/O bits	30

Table 5.2.1: Generic Resource Utilization Report of Sharpening and Noise removal model

The resource utilization for this operation on a targeted FPGA after optimization has been shown below.

Resources	Numbers
Logic Utilization (in ALMs)	793/9430 (8%)
Total registers	1759
Total pins	30/128 (23%)
Total block memory bits	65919/1,802,240 (4%)
Total DSP blocks	2/25(8%)
Combinational ALUT usage for logic	1227

Table 5.2.2: Resource Utilization Report of Sharpening and Noise removal model by Quartus Prime Lite

5.3 Implementing LoG operator

As the LoG operator is basically an improvement over Sobel operator for this proposed algorithm, we have used LoG operator for the edge detection operation. The implementation of this operator has totally been done by using MATLAB script as there is no ready-made building blocks for LoG operator in Simulink or its related add-on toolboxes. Although there is one toolbox for edge detection in Simulink offered by Vision HDL Toolbox but there are only three edge detectors which are Sobel, Prewitt and Roberts. So, if the designer wants to use the same methodology for implementing LoG in Simulink as Local Laplacian and Sharpenig, the options are limited up to only three edge detection techniques.

So, for implementing LoG on MATLAB we have two choices which are either to use inbuilt function for LoG operator or to write a script which performs the LoG operation. But, unfortunately, the MATLAB command for LoG operator has not been supported by HDL coder for the code generation. So, we were forced to write a special script and test bench for implementing this operation. So, giving an overview about the MATLAB implementation, refer to the image shown below in figure.

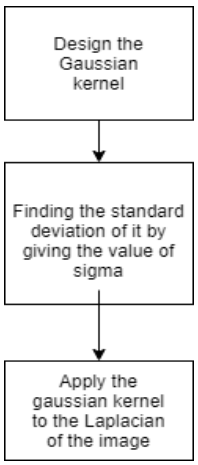

Fig. 5.3.1: Overview of LoG implementation

First of all, the Gaussian kernel was generated and applied standard deviation to that kernel as LoG is a second order derivative of the image. On the other hand, the Laplacian of the image is performed by using Laplacian kernel. At the end, the Gaussian smoothening filter(kernel) is applied to the Laplacian of the image to increase the sensitivity to noise by performing the convolution. The image shown below refers to the MATLAB script for performing the operations mentioned in this section.

```
16
17 -  I = double(img);
18 -  sz = double(4);
19 -  [x,y] = meshgrid( -sz : sz, -sz : sz );
20 -  M = size(x,1) - 1;
21 -  N = size(y,1) - 1;
22 -  exp_comp = -(x .^ 2 + y .^ 2)/(2 * sigma * sigma);
23 -  kernel = cordiccexp(real(exp_comp)/(2 * pi * sigma * sigma));
24 -  output = zeros(size(I));
25 -  I = double(padarray(I, [sz sz]));
26 -  for i = 1: size(I,1) - M
27 -      for j  = 1 : size (I, 2) - N
28 -          Temp = I(i : i + M, j : j + M) .* kernel;
29 -          output(i,j) = real(sum(Temp(:)));
30 -      end
31 -  end
32    %output = unit8(output);
33
```

Fig. 5.3.2: MATLAB script for LoG operation

After debugging the script the testbench for LoG was created and then in HDL coder two files were imported as HDL coder also accepts the MATLAB scripts in

specific form of function and testbench pair. Next, the code was converted to fixed point configuration and HDL generation report was generated by code generator and implemented on generic FPGA which has been shown below.

Resources	**Numbers**
Multipliers	3
Adders/Subtractors	35
Registers	512
Total 1-bit registers	2527
RAMs	4
Multiplexers	179
I/O bits	28

Table 5.3.1: Generic Resource Utilization Report of LoG operator

The resource utilization report of LoG operator is almost same as the Sharpening and Removing model since both of the steps performs almost same steps except LoG performs the edge detection and the previous step performs the Sharpening. The MATLAB simulation took 5.2 seconds for LoG operator and the run time was 2.1 seconds which is considerably good.The resource utilization for this operation on a targeted FPGA after optimization has been shown below.

Resources	**Numbers**
Logic Utilization (in ALMs)	601/9430 (6%)
Total registers	1159
Total pins	28/128 (21%)
Total block memory bits	59316/1,802,240 (3.3%)
Total DSP blocks	2/25(8%)
Combinational ALUT usage for logic	994

Table 5.3.2: Resource Utilization Report of LoG by Quartus Prime Lite

5.4 Implementing Masking

Masking is basically a multiplication of two images as we are masking the sharpened image and the smoothened image. So we need to multiply two images pixel wise to implement this operation. For implementing this operation, product building block offered by Vision HDL Toolbox has been used which also can be used for multiplication or division of the scalar and non scalar matrix. Moreover, like the other operations pixel streaming interface has been used. So, the block diagram of the masking implementation is shown below.

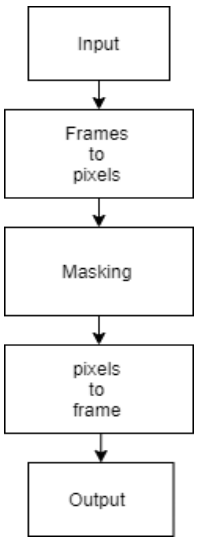

Fig. 5.4.1: Implementation of Masking operation

The mathematical operations such as multiplication, addition. subtraction, division, etc. can be implemented in Simulink so easily since there are ready-made building blocks provided in different toolboxes. Moreover, HDL coder supports these mathematical operations for code generation as well. The runtime for this operation is 2.9 seconds. The generic resource utilization report and the resource utilization for targeted FPGA by Quartus Prime Lite for masking implementation have been shown below.

Resources	**Numbers**
Multipliers	1
Adders/Subtractors	14
Registers	243
Total 1-bit registers	1367
RAMs	1
Multiplexers	82
I/O bits	14

Table 5.4.1: Generic Resource Utilization Report of Masking operator

Resources	Numbers
Logic Utilization (in ALMs)	470/9430 (5%)
Total registers	829
Total pins	16/128 (12.5%)
Total block memory bits	45532/1,802,240 (2.5%)
Total DSP blocks	1/25(8%)
Combinational ALUT usage for logic	458

Table 5.4.2: Resource Utilization Report of masking by Quartus Prime Lite

5.5 Implementing Power Law Transform

In this operation basically, we compare the corrupted image with the original image and then improve the gamma of the corrupted image. For gamma correction, ready-made building block of Vision HDL Toolbox has been used. First, the image is applied to the corruption block which generates the darker image by applying the de-gamma. After that, that image is passed to the pixel-stream Gamma compensation subsystem which improves the gamma of the image. The overview of the implementation of the Gamma Transformation is shown below.

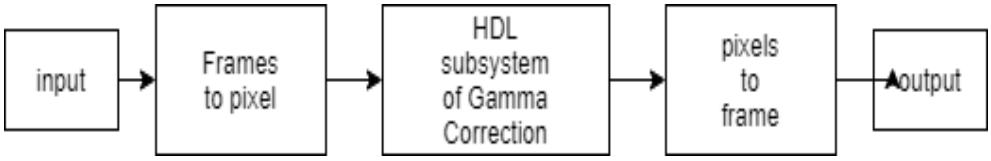

Fig. 5.5.1: Overview of Power Law Transform

HDL subsystem of Gamma Correction block is designed as same as the other steps in this algorithm. For implementing this operation Simulink MATLAB has been used. The generic resource utilization report generated by HDL coder and the resource utilization report generated for a targeted FPGA by Quartus Prime Lite have been shown below. The resource utilization for this operation is the most efficient in

this algorithm as we can see in the resource utilization report. Moreover, the runtime for this operation is 3.1 seconds which also is good.

Resources	**Numbers**
Multipliers	0
Adders/Subtractors	0
Registers	12
Total 1-bit registers	26
RAMs	0
Multiplexers	2
I/O bits	30

Table 5.5.1: Generic Resource Utilization Report of Power Law Transform

Resources	**Numbers**
Logic Utilization (in ALMs)	9/9430 (< 1%)
Total registers	18
Total pins	29/128 (23%)
Total block memory bits	2048/1,802,240 (< 1%)
Total DSP blocks	0/25(8%)
Combinational ALUT usage for logic	0

Table 5.5.2: Resource Utilization Report of Power Law Transform by Quartus Prime Lite

5.6 Quality Estimation Analysis

For estimation of the quality of this improved algorithm an ideal and probably the easiest method from the image quality matrix has been chosen. Peak signal to noise

ratio(PSNR) has been calculated for this algorithm by using MATLAB. As an input image a standard coloured peppers.png image has been used. The original image and the noisy image are passed to the MATLAB script designed to calculate the PSNR, and PSNR is determined for the each steps of the algorithm which are shown below.

Operation	**PSNR**
Local Laplacian	32.20
Laplacian[7]	18.24
LoG	8.89
Sobel[2]	8.69
LoG[2]	8.9
Sharpening	43.08
Sharpening[20]	41.76
Masking	30.07
Adaptive Masking [19]	32.54
Power Law Transform	18.01
Power Law Transform[11]	16.18

Table 5.6.1: Analysis and comparison of quality estimation of the algorithm

In the above table we can observe the different values of PSNR for the different steps of the algorithm. Moreover, their comparison with previous work also has been done to determine the quality of the improved image enhancement algorithm.
To begin with, PSNR for local Laplacian is 32.20 which is quite impressive than the PSNR for Laplacian filter which is 18.24. Moreover, we have proposed to use a Local Laplacian instead of Laplacian to make this algorithm useful for coloured images as well. So quality wise Local Laplacian is better than the Laplacian filter. Secondly, as the improvement of Sobel in edge detection is chosen as LoG in this thesis, with performing efficiently LoG offers a better PSNR as well. Although the PSNR is better

in [2] when it comes to implementing LoG. Additionally, Sharpening operation of the proposed algorithm has a better PSNR than [20] since we have used noise removal block offered by MATLAB in the designing the sharpening model. Apart from that, the PSNR achieved for Masking and Power Law transform are almost similar to the PSNR achieved in [19] and [11] respectively.

5.7 Resource Utilization Analysis

As resource utilization plays an important role in chip cost and power consumption, we ought to choose a model that offers a better resource utilization. Since in this algorithm we have aimed to overcome a performance problem of a Sobel operator and the limitation of a Laplacian operator we have compared the resource utilization tables for those steps in this thesis. We have used "Cyclon V: 5CEBA2F17C6" FPGA as a targeted FPGA and for synthesis we have used Quartus Prime Lite.

Resources	**LoG on Cyclon V**	**Sobel on Cyclon II by[8]**
Logic Utilization (in ALMs)	601/9430	2608/33216
Total registers	1159	3476
Total block memory bits	59316/1,802,240	49152/483840
Combinational ALUT usage for logic	994	2438

Table 5.7.1: Resource utilization comparison for edge detection techniques

As we can see in above table, the hardware resource utilization of the methods used for edge detection are compared. In this thesis we have proposed the usage of LoG in edge detection over sobel to overcome the performance problem and even when it comes to implementing it on FPGA, LoG seems more efficient than Sobel. Here, in [8] Cyclone II : EP2C35F672C8 has been used. They also have used Quartus Prime for synthesis and coloured image as an input.

Resources	Local Laplacian on Cyclon V	Laplacian on Zynq 7000 by [25]
Total registers	10450	13514
Total DSP blocks	2/25	21 /900

Table 5.7.2: Resource utilization comparison for Local Laplacian and Laplacian

In above table we can observe the comparison of fast local laplacian and Laplacian operators. in [8] authors have used Zynq FPGA for implementation so we cannot directly compare every resource utilization parameters with each other but they seem somewhat similar to each others. Moreover, Fast Local Laplacian is an improvement of Laplacian since it also processes the coloured images efficiently.

5.8 Visual Interpretability Analysis

To determine the visual interpretations achieved by this improved image enhancement method we applied this method to an X-ray image size of 1000 x 1092p such as [17]. The simulated images are shown below step wise.

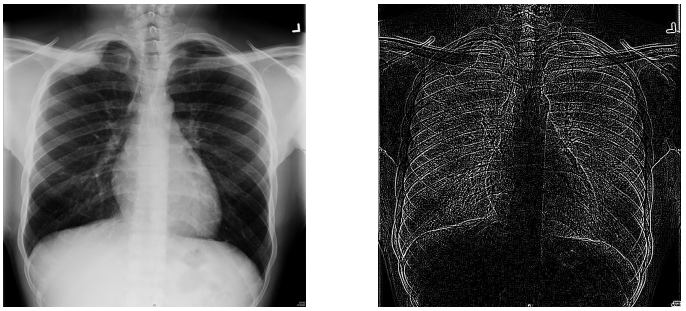

Fig. 5.8.1: X-ray image of human body(left) [12], Laplacian of the image(right)

In Figure 5.8.1 the laplacian of the image taken as an input has been shown. we used the negative Lapacian mask to highlight the details which are visible in forms of edges. Figure 5.8.2 shows the images resulted after sharpening operation and edge detection. Here, for the edge detection the improvement of Sobel has been used. We get overall sharpened image after the sharpening but after LoG, the edges become prominent as we can observe in the image.

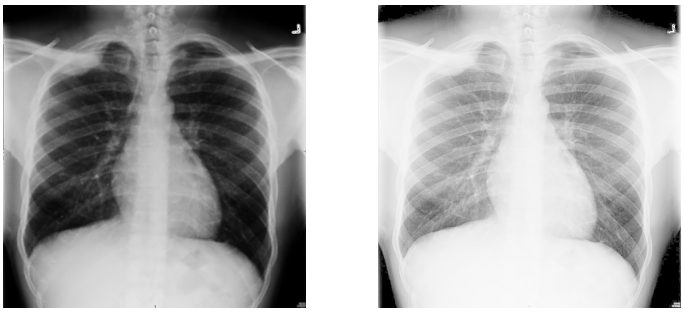

Fig. 5.8.2: Sharpened image by addition of two images(left), LoG of the image(right)

In Figure 5.8.3 the smoothening of the LoG of the image by the Wiener filter and the mask performed by multiplying the sharpened and the smoothened image has been shown. The smoothened image seems a bit blurred as the we applied Gaussian noise to it but the masked image seems like a perfect mixture of a shapening and smoothening.

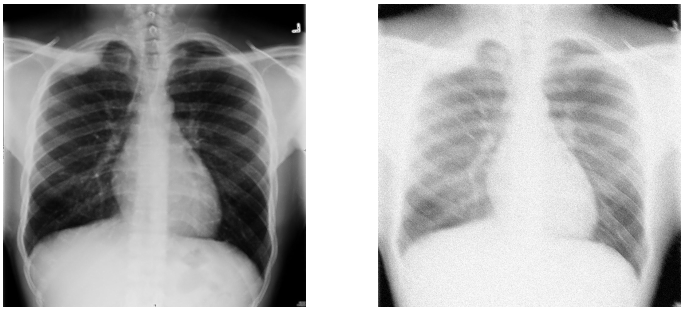

Fig. 5.8.3: Smoothened image by filtering(left), masked formed by multiplication(right)

At last, in Figure 5.8.4 a sharpened image for more effective analysis by adding the sharpened and the mask has been shown along with the gamma transformed image. Here, the value of γ has been chosen to be 0.5. Gamma transform has been used to increase the dynamic range of the image and the details of the sharpened image can be observed containing the significant sharpness and it provides an ease for observation.

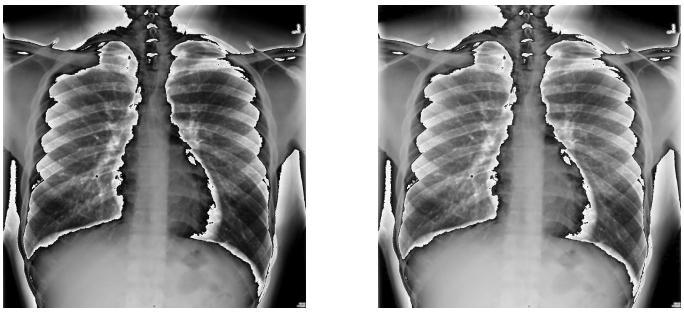

Fig. 5.8.4: Sharpened image by second addition(left), gamma transformed image(right)

Regarding overcoming the limitations of a Laplacian operator, in figure 5.8.5 the Laplacian filtered image and the Local Laplacian filtered image have been shown. We have tested a standard "peppers.png" coloured image and we can see the result. Local Laplacian works just fine with the coloured images whereas Laplacian is only limited to the grey scale images.

Fig. 5.8.5: Laplacian filtered image(left), Local Laplacian filtered image(right)

CHAPTER 6

Conclusion And Future Work

6.1 Conclusion

In this thesis, we addressed the performance problem of Sobel and the limitations of Laplacian operator for image enhancement algorithms and we overcome those problem by offering the better solution for edge detection and information extraction as LoG over Sobel and Fast local Laplacian over Laplacian respectively. We talked in detail about image enhancement algorithm and their alternatives in the initial chapters. After improving the image enhancement algorithm we extended the scope of this algorithm by making it useful for coloured images as well. After that in the following chapters at the implementation part we found the new problem of implementing this algorithm on a parallel hardware for faster and cheaper execution as there are several approaches to implement image enhancement algorithms on FPGA. We discussed every approaches in depth along with their advantages and disadvantages.

In the later sections, we did a detailed analysis of implementing an improved algorithm proposed in this thesis. Moreover, we discussed the methods we used to overcome some problems during implementation part such as feeding an image to FPGA as an input and achieving parallelism. After that, we performed high level synthesis by using MATLAB, Simulink and other additional related tools such as Vision HDL Toolbox and Computer Vision Toolbox. To realize the synthesizability of the HDL script generated by high level synthesis we used Quartus Prime Lite. We implemented the proposed algorithm on a targeted FPGA by using Quartus and performed the timing and hardware resource utilization analysis. Apart from that

for quality estimation we determined the peak noise to signal ratio of each step and compared with the previous approaches and came to the conclusion that the improved algorithm performs efficiently in terms of the image quality. Additionally, we benchmark the resource utilization of our model with existing approaches. Although the approach of implementing the Laplacian is different than our approach we can say that our model of Local Laplacian match the resource utilization with it and performs better at processing the coloured images. The area of applications for this algorithms are computer vision applications, digital cameras, space and military, medical application, etc.

6.2 Future Work

While implementing and analyzing our design models, we restrict them to the resource utilization and quality estimation. In the future for faster approaches and achieving a better spatial or temporal parallelism, the usage of multiprocessors would be beneficial such as Zynq offers FPGA with GPU. Moreover Xilinx design suit would be helpful in taking power consumption aspects into consideration and their optimization. Apart from that, as our improvements for this algorithm are the most viable solutions, they can be implemented more cost effectively on a parallel hardware in future to make them feasible for other applications.

REFERENCES

[1] AlAli, M. I., Mhaidat, K. M., and Aljarrah, I. A. (2013). Implementing image processing algorithms in FPGA hardware. In *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pages 1–5.

[2] Ansari, M., Kurchaniya, D., and Dixit, M. (2017). A Comprehensive Analysis of Image Edge Detection Techniques. *International Journal of Multimedia and Ubiquitous Engineering*, 12:1–12.

[3] Bailey, D. G. (2015). The advantages and limitations of high level synthesis for FPGA based image processing. In *Proceedings of the 9th International Conference on Distributed Smart Cameras*, ICDSC '15, pages 134–139, New York, NY, USA. Association for Computing Machinery.

[4] Bhatti, F., Greiner, T., Heizmann, M., and Ziebarth, M. (2017). A new FPGA based architecture to improve performance of deflectometry image processing algorithm. In *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, pages 559–562.

[5] Chervyakov, N., Lyakhov, P., and Nagornov, N. (2020). Analysis of the Quantization Noise in Discrete Wavelet Transform Filters for 3D Medical Imaging. *Applied Sciences*, 10(4):1223.

[6] Chiuchisan, I. (2013). Implementation of medical image processing algorithm on reconfigurable hardware. In *2013 E-Health and Bioengineering Conference (EHB)*, pages 1–4.

[7] Dahiya, S. (2019). Comparative Analysis of Different Image Enhancement Techniques. *International Journal of Modern Electronics and Communication Engineering*, 7:7–13.

[8] Dash, P. K., Pujari, S., and Nayak, S. (2014). Implementation of edge detection using FPGA model based approach. In *International Conference on Information Communication and Embedded Systems (ICICES2014)*, pages 1–6.

[9] Di Zenzo, S. (1986). A note on the gradient of a multi-image. *Computer Vision, Graphics, and Image Processing*, 33(1):116–125.

[10] Faragallah, O. S., El-Hoseny, H., El-Shafai, W., El-Rahman, W. A., El-Sayed, H. S., El-Rabaie, E.-S. M., El-Samie, F. E. A., and Geweid, G. G. N. (2021). A Comprehensive Survey Analysis for Present Solutions of Medical Image Fusion and Future Directions. *IEEE Access*, 9:11358–11371. Conference Name: IEEE Access.

[11] Fatahbeygi, A. and Akhlaghian Tab, F. (2019). A highly robust and secure image watermarking based on classification and visual cryptography. *Journal of Information Security and Applications*, 45:71–78.

[12] Finlay, K. (2017). Curious Kids: How do x-rays see inside you?

[13] Gribbon, K. T., Bailey, D. G., and Johnston, C. T. (2005). Design Patterns for Image Processing Algorithm Development on FPGAs. In *TENCON 2005 - 2005 IEEE Region 10 Conference*, pages 1–6. ISSN: 2159-3450.

[14] Hai, J. C. T., Pun, O. C., and Haw, T. W. (2015). Accelerating video and image processing design for FPGA using HDL coder and simulink. In *2015 IEEE Conference on Sustainable Utilization And Development In Engineering and Technology (CSUDET)*, pages 1–5.

[15] HUSEJKO, M., EVANS, J., and SILVA, J. (2015). Investigation of High-Level Synthesis tools' applicability to data acquisition systems design based on the CMS ECAL Data Concentrator Card example. *Journal of Physics: Conference Series*, 664:082019.

[16] Ioan, A. D. (2010). Designing an optimal single chip FPGA video interface for embedded systems. In *2010 3rd International Symposium on Electrical and Electronics Engineering (ISEEE)*, pages 58–63.

[17] Islam, S. M. and Mondal, H. S. (2019). Image Enhancement Based Medical Image Analysis. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–5.

[18] Johnston, C. (2005). Implementing Image Processing Algorithms on FPGAs.

[19] Kadam, C. and Borse, S. B. (2017). An Improved Image Denoising Using Spatial Adaptive Mask Filter for Medical Images. In *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, pages 1–5.

[20] Lian, J. (2019). Image Sharpening with Optimized PSNR. In *2019 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 106–110. ISSN: 2642-6471.

[21] Maini, R. and Aggarwal, H. (2010). A Comprehensive Review of Image Enhancement Techniques. *Journal of Computing*, 2.

[22] Metkar, A., Maroo, M., Sawant, A., Singh, V., and Mhatre, S. (2020). Hardware Implementation of Image Processing Algorithms on FPGA. SSRN Scholarly Paper ID 3568140, Social Science Research Network, Rochester, NY.

[23] Muñoz, D., Sanchez, D., Llanos, C., and Ayala-Rincon, M. (2010). Tradeoff of FPGA Design of a Floating-point Library for Arithmetic Operators. *Journal of Integrated Circuits and Systems*, 5.

[24] Nosrat, A. and Seifi Kavian, Y. (2012). Hardware Description of Multi-Directional Fast Sobel Edge Detection Processor by VHDL for Implementing on FPGA. *International Journal of Computer Applications*, 47:1–7.

[25] Onat, E. (2017). FPGA implementation of real time video signal processing using Sobel, Robert, Prewitt and Laplacian filters. In *2017 25th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4.

[26] Padmappriya, S. and Sumalatha, K. (2018). Digital Image Processing Real Time Applications. *International Journal of engineering Science Invention*, pages 46–51.

[27] Qasaimeh, M., Zambreno, J., Jones, P. H., Denolf, K., Lo, J., and Vissers, K. (2019). Analyzing the Energy-Efficiency of Vision Kernels on Embedded CPU, GPU and FPGA Platforms. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 336–336. ISSN: 2576-2621.

[28] Qiang, Z., He, L., Chen, Y., Chen, X., and Xu, D. (2019). Adaptive fast local Laplacian filters and its edge-aware application. *Multimedia Tools and Applications*, 78(1):619–639.

[29] Raji, A., Thaibaoui, A., Petit, E., Bunel, P., and Mimoun, G. (1998). A gray-level transformation-based method for image enhancement. *Pattern Recognition Letters*, 19(13):1207–1212.

[30] Shandilya, R. and Sharma, R. K. (2017). FPGA implementation of image enhancement technique for Automatic Vehicles Number Plate detection. In *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, pages 1010–1017.

[31] Sharumathi, K. and Priyadharsini, R. (2016). A survey on various image enhancement techniques for underwater acoustic images. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 2930–2933.

[32] Sree, V. K. and Rao, P. S. (2013). Hardware implementation of enhancement of retinal fundus image using Simulink. In *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, pages 239–244. ISSN: 2159-2160.

[33] Tanriverdi, F., Schuldt, D., and Thiem, J. (2018). Hyperspectral Imaging:

Color Reconstruction Based on Medical Data. In *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pages 194–199.

[34] Tzeng, J. and Nguyen, T. Q. (2009). Image enhancement for fluid lens camera based on color correlation. *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, 18(4):729–739.

VITA AUCTORIS

NAME:	Prit Ghanshyambhai Patel
PLACE OF BIRTH:	Nadiad, India
YEAR OF BIRTH:	1997
EDUCATION:	Higher Secondary Diploma – Science Stream, New English High School, Nadiad, Gujarat, India, 2015
	G. H. Patel College of Engineering and Technology (GCET), B.E. in Electronics and Communication Engineering, Anand, Gujarat, India, 2019
	University of Windsor, MASc in Electrical and Computer Engineering, Windsor, Ontario, Canada, 2021