

Image Processing Using Digital Programming on FPGA

by

HARDI KACHCHHI

202111071

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY

in

INFORMATION AND COMMUNICATION TECHNOLOGY

to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY



June, 2023

Declaration

I hereby declare that

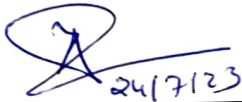
- i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,
- ii) due acknowledgment has been made in the text to all the reference material used.



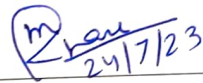
Hardi Kachchhi

Certificate

This is to certify that the thesis work entitled IMAGE PROCESSING USING DIGITAL PROGRAMMING has been carried out by HARDI KACHCHHI for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my/our supervision.



Dr. Yash Agrawal
Thesis Supervisor



Dr. Manish Khare
Thesis Co-Supervisor

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Dr. Yash Agrawal and Dr. Manish Khare, whose expertise played an invaluable role in shaping the research questions and methodology. Their insightful feedback and guidance challenged me to refine my thinking and elevate the quality of my work.

I would like to express my gratitude to the DA-IICT institute providing me the systems required during my research work and staff members for coordinating accordingly.

Furthermore, I would like to extend my heartfelt appreciation to my parents, whose wise counsel and unwavering support have been a constant source of strength. Their presence and understanding have been invaluable throughout this journey. Lastly, I would like to acknowledge the contributions of my friends, whose engaging discussions and joyful distractions provided a much needed relief from my research and helped me maintain a healthy work-life balance. Without the collective support of these individuals, completing this dissertation would have been difficult.

Contents

Abstract	vi
List of Principal Symbols and Acronyms	viii
List of Tables	ix
List of Figures	x
1 Introduction	1
1.1 Motivation	1
1.2 Why FPGA?	2
1.2.1 GPU vs CPU vs FPGA: A Comparative Overview	3
1.3 Organization of thesis	5
2 Image Processing	7
2.1 What is a digital image?	7
2.2 Color Models and Color Spaces	7
2.3 Types of Image Processing	11
2.3.1 Point processing	11
2.3.2 Neighbourhood Processing	12
2.4 Kernel	14
2.5 Filtering	14
2.6 Filters	16
2.6.1 Gray Scale Image	17
2.6.2 Sobel filter	17
2.6.3 Sobel X filter	17
2.6.4 Sobel Y filter	18
2.6.5 Gaussian filter	18
2.6.6 Prewitt X filter	18
2.6.7 Prewitt Y filter	19
2.6.8 Laplacian filter	19

3	Verilog HDL Implementation of Image Processing Algorithms	21
3.1	Design Challenges	21
3.2	Design Flow	21
3.3	Software Specifications	22
3.4	Complete Architecture	23
3.4.1	Row Buffer	23
3.4.2	Control Block	25
3.4.3	Multiply and Accumulate (MAC)	25
3.4.4	FIFO	26
3.5	Simulation Results	26
3.5.1	RTL Schematic	26
3.5.2	Output Images	28
3.5.3	Output Waveform	30
3.5.4	Utilized Resources	30
3.6	Conclusions	31
4	Proposed Method	32
4.1	COE file	32
4.2	Block RAM Generator	33
4.3	Multiply and Accumulate	34
4.3.1	Padding	35
4.4	VGA Controller	36
5	Implementation of the Proposed method	38
5.1	Hardware and Software Used	38
5.1.1	Hardware Required	38
5.1.2	Software Required	38
5.2	Complete Architecture of Proposed Method	39
5.3	COE file generation	40
5.4	Filters Implementation	41
5.5	Block RAM Specifications	42
5.6	VGA Specifications	42
5.7	Flow of Proposed Method	42
5.8	Hardware Setup	44
6	Simulation and Hardware Results	45
6.1	Output Waveform of the Proposed Method	45
6.2	FPGA Implementation Results	47

6.3 Utilized FPGA Hardware Resources	48
7 Conclusions	50
References	51

Abstract

Image processing is a way to transform an image into digital form and after that perform some operations on it that helps to improve images for human interpretation and extract useful information from it. It is essential for a wide range of applications. It allows for enhancing and restoring images, extracting features for object recognition, compressing images for efficient storage and transmission, analyzing images for computer vision tasks, enabling medical diagnostics and treatment, and interpreting data from remote sensing.

Field Programmable Gate Array (FPGA) is preferred for image processing due to their parallel processing capabilities, reconfigurability, low latency, energy efficiency, pipelining support, customization options, real-time processing capabilities, and ease of integration. These advantages make FPGAs a powerful tool for implementing high-performance and efficient image processing solutions across various applications.

To implement various filters in Image processing, we have developed a method that performs various edge detection techniques using FPGAs and displaying the image on the monitor through Video Graphics Array (VGA) Controller. Edge detection filters and blurring filters are an indispensable part of Image processing in various fields due to their ability to extract information, enhance visual quality, and enable decision-making based on visual data .

List of Principal Symbols and Acronyms

BMG	Block RAM Generator
bmp	Bitmap
BRAM	Block Random Memory Access
CLB	Congigurable Logic Block
COE	COEfficient
CPU	Central Processing Unit
CYMK	Cyan Yellow Majenta Key-Color
DDR	Double Data Rate
DSP	Digital Signal Processor
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HSV	Hue Saturation Value
IP	Intellectual Property
ISE	Integrated Synthesis Environment
LUT	Look-up Table
LUT	Look-up table
MAC	Multiply and Accumulate

MUX	Multiplexer
RAM	Random Access Memory
RB	Row Buffer
RGB	Red Green Blue
RTL	Resistor Transfer Level
VGA	Video Graphics Array

List of Tables

5.1 Filter Selection 41

List of Figures

1.1	CPU vs GPU vs FPGA : A Comparative Overview	4
2.1	Gray Scale Image	8
2.2	RGB image layers	9
2.3	HSV color space	10
2.4	Point Processing Of Image	12
2.5	Neighbourhood Processing Of Image	13
2.6	Kernel	14
2.7	Filtering Operation of an image	16
3.1	Design Flow 1	22
3.2	Design Flow 2	22
3.3	Complete Architecture of the Design	23
3.4	Filling of Row buffers	24
3.5	Row buffer IP	24
3.6	Control Block	25
3.7	Multiply and Accumulate IP	26
3.8	RTL Schematic of the design	27
3.9	Output Image(1)	28
3.10	Output Image(2)	29
3.11	Output Waveform	30
3.12	Hierarchy of the design	30
3.13	Resource Utilization	31
3.14	Power consumption	31
4.1	The Block diagram of proposed method	32
4.2	Block RAM	33
4.3	VGA Display	36
5.1	Complete Architecture	39
5.2	COE file Generation	40

5.3	COE file	41
5.4	VGA interfacing with FPGA	42
5.5	Flow of Proposed Method (1)	43
5.6	Flow of Proposed Method (2)	43
5.7	Flow of Proposed Method (3)	43
5.8	Hardware Setup	44
5.9	FPGA board	44
6.1	Output Waveform	46
6.2	FPGA Results	47
6.3	Resource Utilization	48
6.4	Power Consumption	48
6.5	Utilization summary	49

CHAPTER 1

Introduction

1.1 Motivation

Image processing is a captivating field that revolves around manipulating data that inherently exists in a two-dimensional form. It covers a wide range of techniques that are utilized in various domains. One prominent application of image processing is image enhancement [1] [2].

Image enhancement is a transformative procedure that aims to enhance an image's visual quality and clarity of image, enabling a better perception and understanding of the essential features. By adjusting various parameters and applying specialized techniques, image enhancement improves visibility, enabling easier identification and analysis of specific elements within the image. This enhanced visual representation not only aids in highlighting relevant details but also facilitates subsequent image-based investigations, such as accurate measurements or precise object classification. Overall, image enhancement is crucial in refining and optimizing images to extract valuable information and facilitate more effective image-based tasks [3].

One of the major steps in image enhancement is the filtering of images. Image filters are essential tools used in various ways to analyze and manipulate images. The fundamental and elementary applications of filters are edge detection and smoothing of images. Edge detection plays a crucial role in image segmentation and feature extraction. By applying edge detection techniques, it becomes possible to identify and highlight the boundaries or edges of objects within an image. Image smoothing, also known as image blurring, is a technique that reduces noise and removes fine details from an image, resulting in a smoother and more visually appealing appearance. This information can then be used to separate different regions of the image, extract important features, or enhance the overall visual quality [4].

1.2 Why FPGA?

FPGAs are becoming more prevalent in image processing applications, particularly in real-time embedded scenarios where low latency and power efficiency are crucial factors. Image processing using FPGAs offers several advantages over other computing platforms. Here are some reasons why FPGA-based image processing is beneficial [5] [6]:

- **Parallel Processing:** FPGAs excel at parallel processing, allowing them to perform multiple computations simultaneously. This is particularly advantageous for image processing, as images consist of a large number of pixels that can be processed independently. FPGAs can process multiple pixels in parallel, leading to significant speed improvements compared to sequential processing on CPUs or GPUs.
- **Customizability:** FPGAs are highly customizable hardware devices. They can be programmed and reconfigured to implement specific image processing algorithms or even entire image processing pipelines. This flexibility allows developers to optimize their algorithms for specific applications and achieve higher performance compared to general-purpose processors.
- **Low Latency:** FPGA-based image processing can achieve lower latency because FPGAs offer direct access to hardware resources without the need for intermediate layers or complex memory hierarchies. This makes them suitable for real-time applications, such as video processing, where low latency is crucial.
- **Energy Efficiency:** FPGAs are known for their energy efficiency. By implementing image processing algorithms directly in hardware, FPGAs can achieve high computational throughput while consuming lesser power compared to traditional CPUs or GPUs. This is particularly important in applications where power consumption is a limiting factor, such as portable devices or embedded systems.
- **Hardware Acceleration:** FPGAs can be used as hardware accelerators alongside CPUs or GPUs. They can offload computationally intensive tasks in image processing algorithms, allowing the CPU or GPU to focus on other tasks. This combination of hardware acceleration can significantly speed up image processing tasks and improve overall system performance.

- **Real-Time Adaptability:** FPGAs offer the ability to reconfigure their hardware on the fly. This feature allows real-time adaptation to changing image processing requirements or dynamic environments. It can be particularly useful in applications where algorithms need to be adjusted or updated based on real-time feedback. Overall, FPGA-based image processing provides high performance, low latency, energy efficiency, and customization capabilities, making them an attractive choice for various applications ranging from computer vision and surveillance to medical imaging and industrial automation.

1.2.1 GPU vs CPU vs FPGA: A Comparative Overview

A CPU is a versatile processor that executes instructions and performs calculations for various tasks in a computer system whereas a GPU is a specialized processor primarily designed for handling and accelerating graphical computations. Initially developed for rendering graphics in video games, modern GPUs have evolved to become highly parallel processors capable of performing complex mathematical calculations. On the other hand, FPGA is a programmable integrated circuit that can be configured to perform specific functions. Unlike CPUs and GPUs, which are fixed-function processors, FPGAs can be reprogrammed to implement custom digital logic circuits and algorithms. This flexibility makes FPGAs suitable for a wide range of applications, including digital signal processing, real-time data processing, cryptography, and hardware acceleration for specific tasks [7] [8].

In certain applications, such as time-sensitive analysis, fast processing is crucial, and regular CPUs are not capable of performing calculations quickly enough to meet the requirements. When it comes to training small neural networks with a limited dataset, CPUs can provide certain cost advantages initially. Nonetheless, there is a benefit to using CPU-based applications in terms of power consumption. When compared to GPU configurations, CPUs tend to offer better energy efficiency. So, while CPUs may be slower in training neural networks, they can be a more power-efficient choice in certain scenarios. [9].

Hardware accelerators like DSPs, FPGAs, and GPUs are used to speed up computation time for algorithms [9]. FPGAs possess the remarkable capability to concurrently host multiple functions, allowing for dedicated portions of the chip to be assigned to specific tasks. This distinctive architecture greatly enhances both operational efficiency and energy consumption. By incorporating distributed memory directly into the fabric, FPGAs bring the memory closer to the processing com-

ponents, resulting in reduced latency. Notably, this arrangement can significantly decrease power usage compared to traditional GPU designs. In summary, FPGAs stand out for their ability to optimize performance and energy efficiency by seamlessly integrating memory and processing elements. [10]. The limited size of local memory in GPU groups restricts their ability to execute certain algorithms, making them lesser suitable than FPGAs for some applications. To achieve high performance on GPUs, careful algorithm design is necessary to overcome these limitations. Although programming tools for GPUs have been developed, achieving optimal performance on GPUs, CPUs, and FPGAs remains challenging [10].

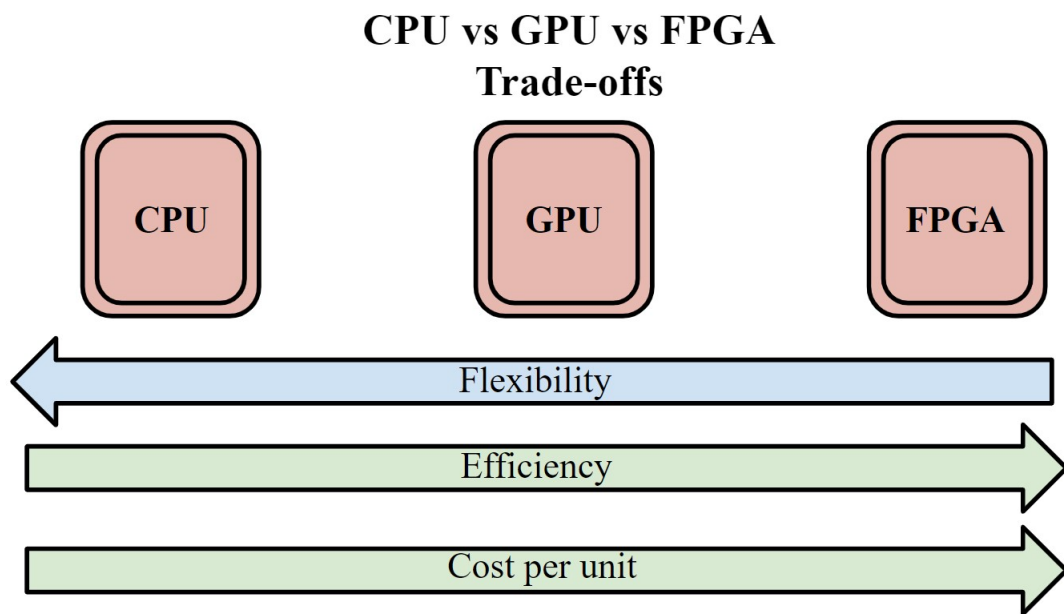


Figure 1.1: CPU vs GPU vs FPGA:A Comparative Overview

Various hardware accelerators are considered and discussed in the below discussed papers for their suitability. Several examples include the utilization of multicore CPUs, FPGAs, and GPUs in the field of experimental mechanics [11]. Furthermore, GPUs are commonly employed for image processing [12] [13], and for image registration, GPUs [14] as well as multicore CPUs and GPUs [15] are often employed. While these papers offer valuable insights, it is important to note that some of them solely focus on GPUs, neglecting the discussion of FPGAs or DSPs. Moreover, the technical details in these papers tend to be specific to particular hardware architectures, limiting their usefulness in directly comparing different hardware accelerators.

In conclusion, CPUs, GPUs, and FPGAs are hardware accelerators with distinct characteristics and advantages. CPUs are versatile processors suitable for a

wide range of tasks, but these may not meet the fast processing requirements of certain applications. GPUs excel at graphical computations and have evolved into highly parallel processors capable of complex mathematical calculations, making them well-suited for training neural networks. However, they can consume more power compared to CPUs. FPGAs, on the other hand, offer the unique advantage of reprogrammability, allowing for the implementation of custom digital logic circuits and algorithms. These excel in optimizing performance and energy efficiency by integrating memory and processing elements, also they are good in decreasing power consumption as compare to GPUs. FPGAs are particularly beneficial for applications that require concurrent execution of multiple functions and where GPU limitations hinder algorithm execution. While GPUs have been extensively studied and employed in various fields, including image processing, some papers tend to overlook the discussion of FPGAs and DSPs. A similar trade-off between all the hardware accelerators can be seen in figure 1.1. Overall, the choice between CPU, GPU, and FPGA depends on the specific requirements and constraints of the application at hand.

1.3 Organization of thesis

The thesis organization is as follows:

The present **Chapter 1** gives the introduction on FPGA, its necessity and organization of the thesis.

Chapter 2 focuses on the fundamentals of image processing, which includes a detailed understanding of image processing, types of image processing, filtering process of images and different types of filters.

Chapter 3 delves into the Verilog HDL implementation of Image Processing filters such as Blur filter and Sobel Edge filter.

Chapter 4 introduces the proposed method for filtering of images using FPGA and VGA along with a block diagram and complete architecture of proposed method.

Chapter 5 delves into the implementation part which is done using verilog HDL language and FPGA board as hardware.

Chapter 6 discusses the simulation results as well as the hardware results obtained by implementing the proposed method.

Chapter 7 concludes all the work implemented, its application and future explorations.

CHAPTER 2

Image Processing

2.1 What is a digital image?

Image processing refers to processing of digital images. A digital image is a representation of an image, denoted as $f(x, y)$, that has been discretized both in coordinates and brightness. It is typically expressed as a 2D integer array or a set of 2D arrays, where each array corresponds to a color band. The discrete values assigned to represent brightness are referred to as gray levels. [16] In this context, each element of the array is referred to as a pixel. The typical size of such an array ranges from a few hundred pixels by a few hundred pixels, and the number of available gray levels generally amounts to several dozens. Consequently, a digital image can be visualized as shown in the equation 2.1 [16]:

$$f(x, y) = \begin{bmatrix} f(1,1) & f(1,2) & f(1,3) & \dots & f(1,N) \\ f(2,1) & f(2,2) & f(2,3) & \dots & f(2,N) \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & \dots & \cdot \\ f(N,1) & f(N,2) & f(N,3) & \dots & f(N,N) \end{bmatrix} \quad (2.1)$$

2.2 Color Models and Color Spaces

The human eye perceives color as light within a relatively limited range of the electromagnetic spectrum. Due to the biology of the eye, it is especially sensitive to red, blue, and green light. By combining these three primary colors in varying proportions, humans are capable of perceiving a wide array of colors.

A color model serves as a geometric or mathematical framework designed to capture and explain the colors perceived by humans. It employs numerical values assigned to specific dimensions within the model to represent the full range of

colors visible to the human eye. By utilizing these numerical values, a color model provides us with a systematic approach for describing, categorizing, comparing, and arranging colors in a meaningful way.

In the context of color representation, a color space serves as a practical implementation of a color model. It defines a specific range of colors that can be achieved using that particular color model. While the color model establishes the relationship between values, the color space assigns a definitive interpretation to those values as actual colors. By combining the principles of the color model and the specifications of the color space, we can accurately represent and reproduce a wide array of colors within a given system or application.

An image consists of one or more color channels determining the intensity or color at a specific pixel location. In its simplest form, each pixel location contains a single numerical value representing the signal strength at that particular point in the image. To convert these numerical values into a visual image, a color map is used. A color map assigns a specific shade of color to each numerical level in the image, creating a visual representation of the data. Following are the color spaces that are widely used:

- **Gray Color space:** In digital photography, computer-generated imagery, and colourimetry, a grayscale image refers to an image where each pixel holds a single sample representing the amount of light present, conveying solely the intensity information. This range is often scaled to 0-255 in the representation of an 8-bit image. These grayscale images, also known as black-and-white or gray monochrome images, consist entirely of various shades of gray. The contrast within these images spans from black, representing the lowest intensity, to white, indicating the highest intensity [17]. Example of a gray scale image is shown in figure 2.1.

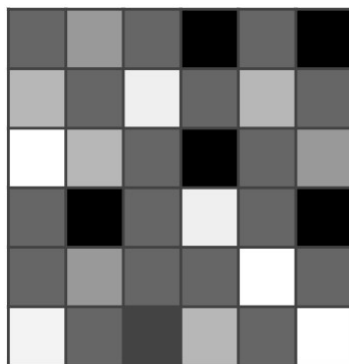


Figure 2.1: Gray Scale Image

- **RGB color space:**

The RGB color space can be seen as a three-dimensional R, G, and B, as depicted in Figure 2.2. Each axis has a range from 0 to 1, although for practical purposes, this range is often scaled to 0-255 in the representation of a 24-bit image, with one byte that is equal to 8 bits per color channel [17] [18].

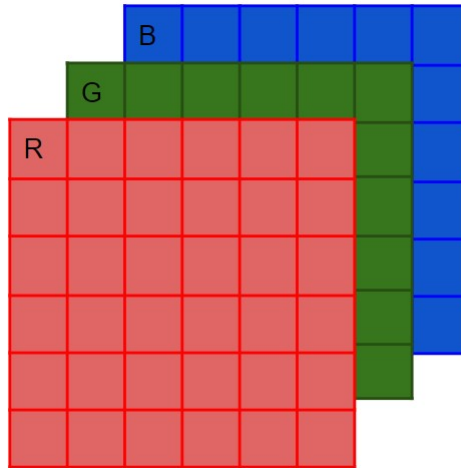


Figure 2.2: RGB image layers

- **HSV color space:**

The HSV color model is a method used to define colors based on three fundamental characteristics: hue, saturation, and Value (luminance or Intensity). The Color space of hsv model is shown in the figure 2.3. This image is generated from Matlab for understanding purpose [19].

- **Hue:**

It represents the color itself and corresponds to color names like red or yellow. It is measured on a scale from 0 to 360, indicating its position on the standard color wheel [19].

Saturation:

It refers to the purity or intensity of a color. A higher saturation value indicates a more vibrant and vivid color, while lower values result in more muted shades. Saturation is typically expressed as a percentage ranging from 0% to 100% [19].

Value:

It also known as brightness, represents the overall lightness or darkness of a color. It is measured on a scale from 0% to 100%, where 0% represents absolute black and 100% represents pure white [19].

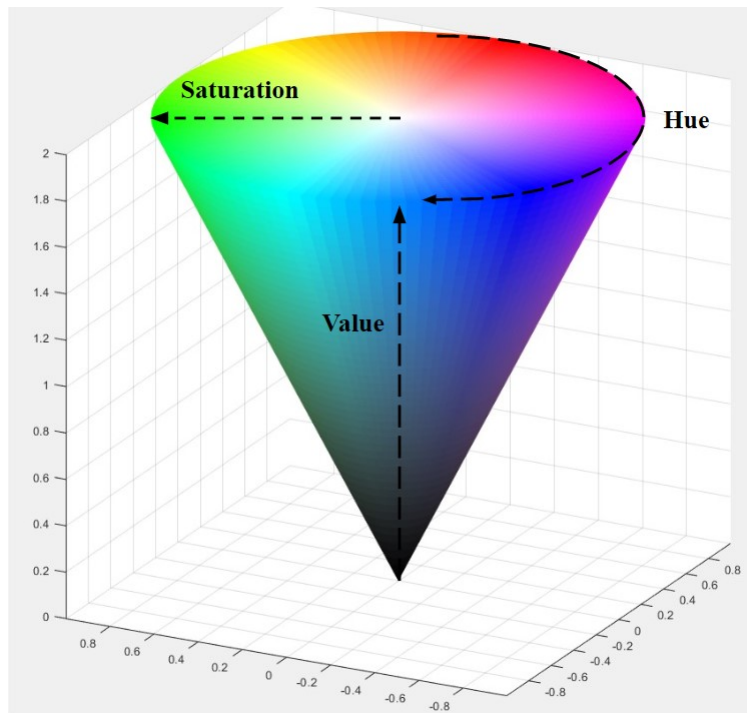


Figure 2.3: HSV color space

By combining these three components, the HSV color model provides a comprehensive way to describe and specify colors.

- **CYMK color space:**

Color spaces that are based on the CYM (cyan, yellow, magenta) color model are classified as subtractive. In this model, cyan, yellow, and magenta represent the primary components. The prominent color space derived from the CYM model is CYMK, where the "K" signifies the key color, which is black [20] [21]. The subtractive color theory, which forms the basis of CYM, states that different levels of cyan, magenta, and yellow absorb or subtract a portion of the spectrum from the white light that illuminates an object. The color of any random object is determined by the lights that are not absorbed. In the CYMK color space, black is introduced to compensate for the interaction of the three primary colors on white paper. CYMK is predominantly used in color printers and similar output devices [21].

The most commonly used color map is grayscale, where different shades of gray, ranging from black (representing zero) to white (representing the maximum value), are assigned according to the signal strength. Grayscale is especially suitable for intensity images, which convey only the intensity of the signal as a single value at each point in the image [17].

Despite the prevalence of color images in our daily lives. This preference for grayscale is rooted in several reasons. Firstly, many image processing operations can be easily extended to color images by applying them independently to each color channel. This approach allows for efficient processing and manipulation of color information. Secondly, a significant amount of visual information can be effectively conveyed through grayscale representations, making color unnecessary for certain analysis and extraction tasks [16].

This can be observed in the historical acceptance of black and white television and the enduring popularity of black and white photography. Lastly, the development of image processing techniques primarily occurred during a time when color digital cameras were costly and not as widely accessible. As a result, algorithms were established and optimized based on the available grayscale image data. Consequently, these techniques have become well-established in the field of image processing [16].

2.3 Types of Image Processing

2.3.1 Point processing

To illustrate this, let's consider the scenario of adjusting the brightness in a movie. The input image would be the actual movie stored on the DVD, while the output image would be the visual representation displayed on the TV screen. If we have an input image denoted as $f(x, y)$ and we aim to obtain a modified version known as the output image that will be denoted by $g(x, y)$.

Now, point processing comes into play. It is a specific type of operation performed on each pixel of the image independently, without taking into account the values of its neighboring pixels. This means that when calculating the new value of a pixel in the output image $g(x, y)$, only the corresponding pixel in the input image $f(x, y)$ is considered. The values of nearby pixels do not influence the computation. As a result, this operation is named point processing because it focuses solely on individual pixels as isolated points which can be seen in figure 2.4 [22].

Point processing operations can vary depending on the desired manipulation of the image. Common examples include adjusting brightness, contrast, or applying color transformations. For instance, to change the brightness of an image, each pixel's intensity value in the output image is determined based solely on the intensity value of the corresponding pixel in the input image [17] [22].

The benefit of point processing lies in its simplicity and efficiency. Since it

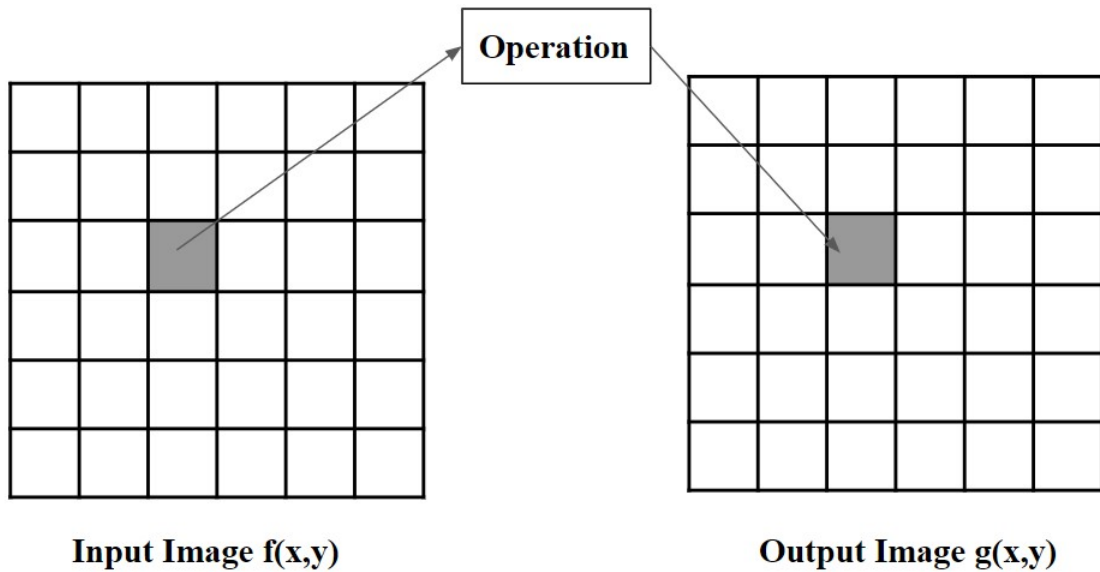


Figure 2.4: Point Processing Of Image

considers each pixel independently, it allows for straightforward calculations and enables real-time processing of images and videos. However, it's important to note that point processing may not be appropriate for tasks that demand considering the relationships and interactions between neighbouring pixels, such as edge detection or noise reduction, which typically involve more complex processing techniques. Overall, point processing provides a fundamental approach to manipulating images by performing operations on individual pixels independently, disregarding the influence of neighbouring pixels [22].

When using your remote to adjust the brightness, you are effectively altering the value of 'b' in the given equation 2.2 [22].

$$g(x,y) = f(x,y) + b \tag{2.2}$$

If 'b' is greater than zero, the image becomes brighter, while if 'b' is lesser than zero, the image becomes darker. In other words, by modifying the 'b' value, you can control the brightness level of the image displayed on your screen.

2.3.2 Neighbourhood Processing

In image processing, the transformation of an input image into an output image often involves considering the surrounding pixels of each pixel of input image in order to determine its value in the resulting image. This concept is known as neighborhood processing [17] [23].

When performing neighborhood processing, we examine the pixel at a specific position in the input image, denoted as $f(x, y)$, and also take into account the values of its neighboring pixels. The neighborhood typically consists of a defined number of adjacent pixels, such as the immediate neighboring pixels in a square or circular region around the central pixel as shown in figure 2.5 [23].

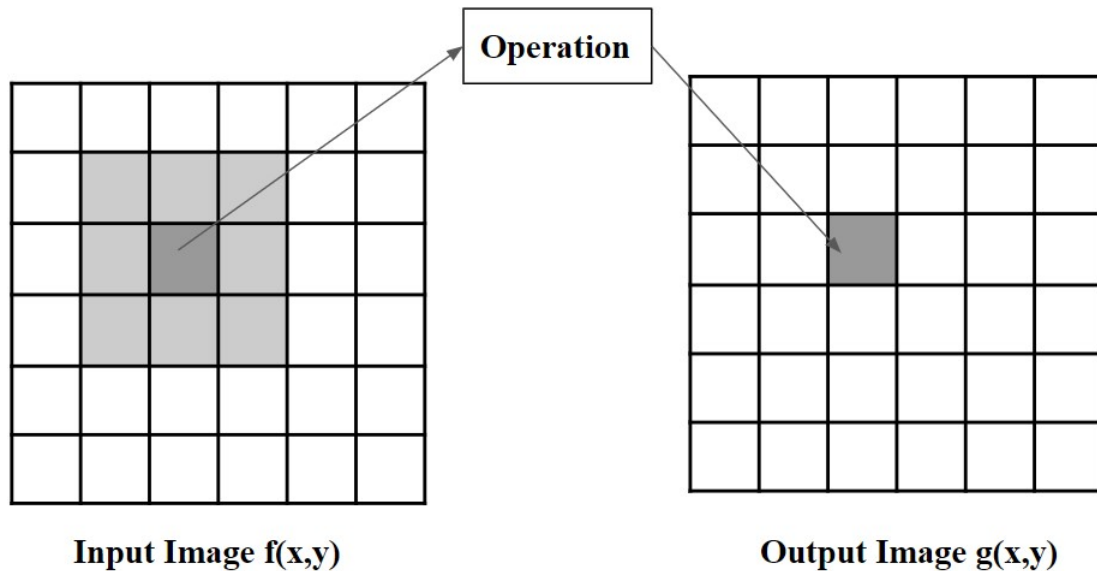


Figure 2.5: Neighbourhood Processing Of Image

The way in which the neighboring pixel values are combined with the central pixel's value to determine the output pixel's value depends on the specific neighborhood processing operation being applied. There are various techniques available, such as averaging the pixel values, applying mathematical filters or kernels, or using complex algorithms to enhance or manipulate the image [23].

By incorporating information from the surrounding pixels, neighborhood processing operations can capture local patterns, gradients, or structures in the image. This allows for various image enhancement, noise reduction, edge detection, or other image manipulation tasks.

Overall, the process of neighborhood processing enables us to transform the input image by considering not only the individual pixel values but also the relationships and context between neighboring pixels. This helps to achieve desired visual effects or extract meaningful information from the image.

2.4 Kernel

A kernel refers to a small matrix or filter used for various operations, such as filtering, convolution, or morphological operations. The kernel is typically a square matrix, but it can also be rectangular or have other shapes. Each element of the kernel contains a numerical weight or value that determines its effect on the neighbouring pixels during the image processing operation. Kernel mostly used are of 3x3 matrix or 5x5 matrix as shown in the figure 2.6 [23].

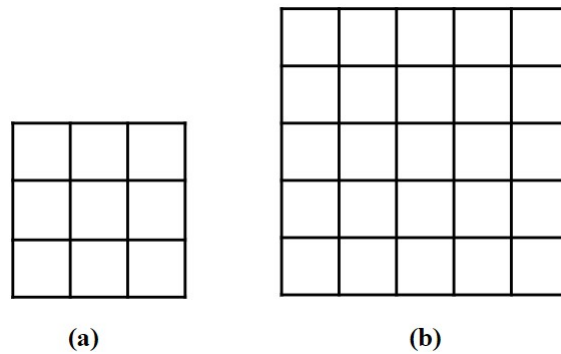


Figure 2.6: Kernel (a) 3x3x Kernel (b) 5x5 Kernel

2.5 Filtering

The combination of Kernel and Function is called filter and the process is called filtering. Filtering is a powerful technique utilized in image processing to bring about modifications or enhancements in an image. This process involves applying various filters to an image, highlighting specific features while eliminating unwanted elements. Filtering operations encompass a range of effects, such as smoothing to reduce noise, sharpening to enhance details, and edge enhancement to make edges more prominent. By employing these filtering techniques, images can be transformed to emphasize desired characteristics or remove distractions, resulting in visually appealing and more informative visual representations. There are two types of filtering: Linear filtering and Non-linear filtering [23].

Linear Filtering:

Linear filtering is a method of image processing where value of each output pixel's value is determined by combining the values of the surrounding pixels in the input image. This combination is achieved through a linear

equation, which assigns weights to each input pixel's value. The output pixel's value is calculated by taking the sum of these weighted values. This type of filtering allows for the modification and enhancement of images by applying various transformations based on the values and relationships of neighbouring pixels. The linear nature of the filtering process enables a wide range of operations, such as blurring, sharpening, and edge detection, to be performed on digital images [23].

Non-linear Filtering:

Non-linear filtering refers to a method of image processing where the value of an output pixel is determined through a non-linear combination of the values of the pixels in the input pixel's neighbourhood. Unlike linear filtering, which uses a linear equation with fixed weights, non-linear filtering involves applying non-linear operations to the pixel values [23] [24].

In non-linear filtering, the value of output pixel's value is not simply a weighted sum of the input pixel values. Instead, it is computed using more complex functions that can account for various factors such as pixel intensity, local contrast, or statistical properties of the neighbouring pixels. These non-linear operations can introduce non-linear relationships and transformations in the resulting image, allowing for different effects and enhancements [23].

In this work, Linear Filtering is used. And filtering involves several steps to process an image [24]:

- 1) A Kernel or filter is positioned over each pixel in the image matrix.
- 2) The filter elements are multiplied by the corresponding elements in the neighbourhood of the pixel. This multiplication is carried out for each element of the filter.
- 3) The products obtained from the multiplication are added together and one output pixel is obtained.

These steps are repeated for every pixel in the image matrix, ensuring the filtering process is applied uniformly across the entire image. The diagram in figure 2.7 shows Linear filtering [23].

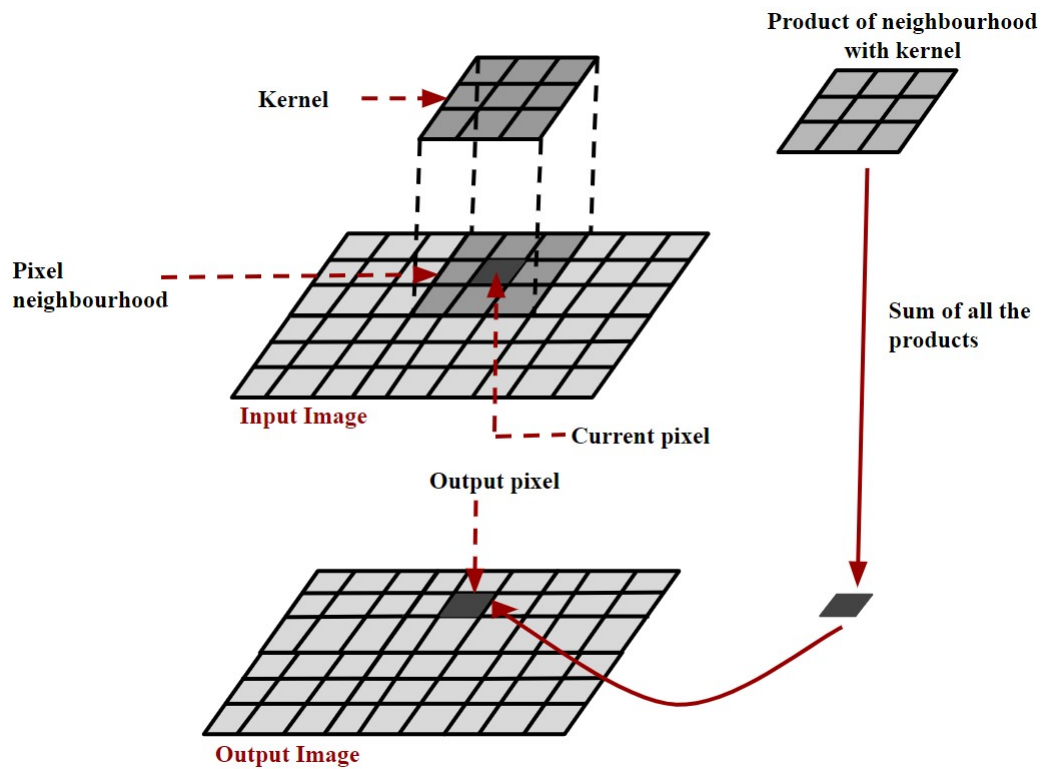


Figure 2.7: Filtering Operation of an image

2.6 Filters

Some very used filters are edge detection filters and smoothing filters. Edge detection is a fundamental operation in image processing that plays a crucial role in various applications. It involves identifying and highlighting the boundaries or edges between different objects or regions within an image. Here are some reasons why edge detection is important:

- 1) Object recognition and segmentation
- 2) Feature extraction
- 3) Image enhancement
- 4) Compression and data reduction
- 5) Object tracking and motion detection
- 6) Edge-based image analysis
- 7) Image understanding and computer vision

A smoothing filter is a digital signal processing technique used to reduce the sharp or abrupt changes in a signal. It is commonly used to remove noise or unwanted variations in data, resulting in a smoother signal representation.

2.6.1 Gray Scale Image

For filtering, conversion from RGB space to grayscale is needed. There are many methods to convert a RGB image into gray scale but the most preferred one is the luminosity method. Researchers have conducted experiments and extensive analysis, leading them to determine the following equation 2.3 to calculate grayscale values:

$$\text{Grayscale} = 0.3 * \text{Red} + 0.59 * \text{Green} + 0.11 * \text{Blue} \quad (2.3)$$

According to the findings, the contribution of the blue color channel should be reduced, while the contribution of the green color channel should be increased in order to obtain the desired grayscale output.

2.6.2 Sobel filter

The Sobel filter, also known as the Sobel operator or edge detection filter, is a commonly used image processing technique to detect edges in an image. It is named after its inventor, Irwin Sobel. The filter works by convolving a small kernel or mask with the input image. The kernel consists of two separate 3x3 matrices, one for detecting horizontal changes (Sobel X) and the other for detecting vertical changes (Sobel Y). The Sobel X matrix emphasizes changes in pixel intensity along the horizontal axis, while the Sobel Y matrix emphasizes changes along the vertical axis. To obtain the final Sobel Filter, the magnitude of the gradients can be computed using the equation 2.4:

$$\text{Sobel} = \sqrt{(\text{SobelX})^2 + (\text{SobelY})^2} \quad (2.4)$$

2.6.3 Sobel X filter

The horizontal Sobel filter is a 3x3 matrix used to detect changes in intensity between adjacent pixels in the same row. The horizontal Sobel filter has values that are positive on the right side of the matrix and negative on the left side of the matrix, which makes it sensitive to edges that transition from dark to light on the right side and from light to dark on the left side. The filter for Sobel X edge detection in equation 2.5.

$$\text{Sobel x filter} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (2.5)$$

2.6.4 Sobel Y filter

The vertical Sobel filter is a 3x3 matrix used to detect changes in intensity between adjacent pixels in the same column. The vertical Sobel filter has values that are positive at the bottom and negative at the top, which makes it sensitive to edges that transition from dark to light at the bottom and from light to dark at the top. The filter for Sobel Y edge detection in equation 2.6.

$$\text{Sobel Y filter} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.6)$$

2.6.5 Gaussian filter

A Gaussian filter is a mathematical algorithm used in image processing to smooth and blur images. The Gaussian filter is commonly used in image processing applications such as noise reduction, edge detection, and feature extraction. It is preferred over other types of filters because it provides a smooth output image that preserves edges and important features while reducing noise. The filter for Gaussian Blur in equation 2.7.

$$\text{Gaussian filter} = \left[\frac{1}{16} \right] \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.7)$$

2.6.6 Prewitt X filter

The horizontal Prewitt filter is a 3x3 kernel that is designed to detect edges that run horizontally across an image. The central column of zeros in the kernel ensures that it is only sensitive to changes in pixel intensity along the horizontal direction. The filter output will be large where there is a strong horizontal edge in the image. The filter for Prewitt X edge detection in equation 2.8.

$$\text{Prewitt X filter} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad (2.8)$$

2.6.7 Prewitt Y filter

The vertical Prewitt filter is also a 3x3 kernel that is designed to detect edges that run vertically in an image. The central row of zeros in the kernel ensures that it is only sensitive to changes in pixel intensity along the vertical direction. The filter output will be large where there is a strong vertical edge in the image. The filter Prewitt Y edge detection in equation 2.9.

$$\text{Prewitt Y filter} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (2.9)$$

2.6.8 Laplacian filter

A 2-D isotropic measure of an image's second derivative is called the Laplacian. Edge detection frequently uses the Laplacian of an image because it shows areas where there are fast changes in intensity. The two variations will be discussed together since the Laplacian is frequently used to minimise noise sensitivity after an image has been first smoothed utilising a method that is similar to a Gaussian smoothing filter.

- **Laplacian 4 Filter:**

The Laplacian 4 filter, also known as the 4-neighbour Laplacian, considers only a pixel's immediate four neighbours (horizontal and vertical directions) in the image. This filter emphasizes changes in intensity along the vertical and horizontal directions, but it does not take into account diagonal edges. The filter for Laplacian edge detection in equation 2.10.

$$\text{Laplacian 4 filter} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (2.10)$$

- **Laplacian 8 Filter:**

The Laplacian 8 filter, also known as the 8-neighbour Laplacian, considers all eight neighbouring pixels (horizontal, vertical, and diagonal directions) of a pixel in the image. The Laplacian 8 filter considers a wider range of directions, including diagonal edges. This makes it more sensitive to edges

in all directions. The filter for Laplacian edge detection in equation 2.11.

$$\text{Laplacian 8 filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (2.11)$$

Therefore, the Laplacian 4 filter is a simpler version that emphasizes vertical and horizontal edges, while the Laplacian 8 filter considers additional diagonal edges. The choice between these filters depends on the specific application and the desired edge detection characteristics.

CHAPTER 3

Verilog HDL Implementation of Image Processing Algorithms

3.1 Design Challenges

FPGAs have lesser memory compared to the memory required by the images for image processing, so there is an issue of memory and to solve that:

- Pure streaming architecture for neighbourhood operations cannot be done since pixels for processing are not consecutive; as studied earlier, it takes its neighbourhood pixels also. So we need to buffer pixels inside the IP before processing it. Buffers are these small memories used for storing one line of the image.
- It is not practical to buffer the entire image simultaneously because, for example, A 512x512 image requires 262144 bytes. A memory inside FPGA can be built using Block RAM or LUTs and flip-flops, but the number of flipflops is quite limited and if CLBs (Configurable Logic Block) are used, then there won't be enough CLBs remaining to implement the remaining logic.
- So, just enough pixels are buffered first. For example, for a 3x3 kernel, three Row buffers are needed to start the image processing.

3.2 Design Flow

The design flow of this method is seen in Figure 3.1. The input image is converted to a digital form of an image in the form of pixels. After that, the pixels are filled in different Row buffers.

Using Multiply and Accumulate, the filtering process is conducted, and we get one output pixel. A similar process is repeated until the whole process is com-

pleted, and we get our final output image. Figure 3.1 show the filtering of one pixel using eight neighbourhood pixels.

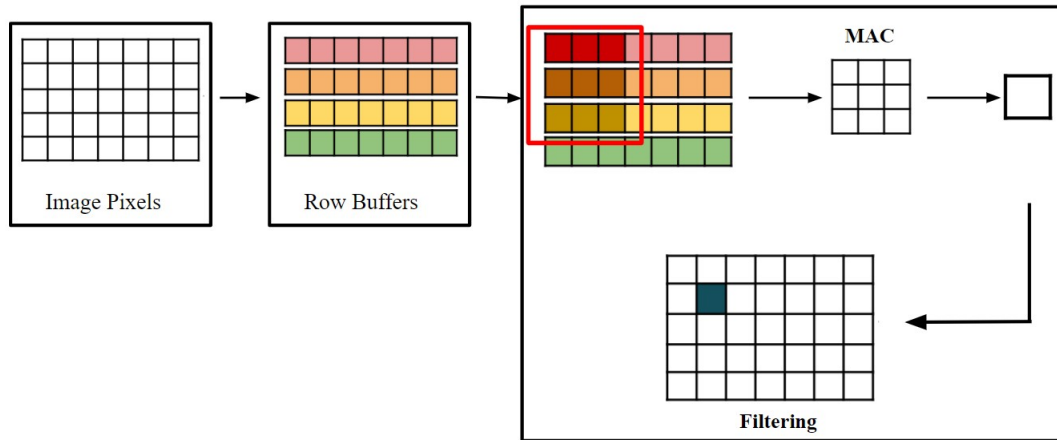


Figure 3.1: Design Flow 1

Figure 3.2 show the filtering of the next pixel after we get our first pixel.

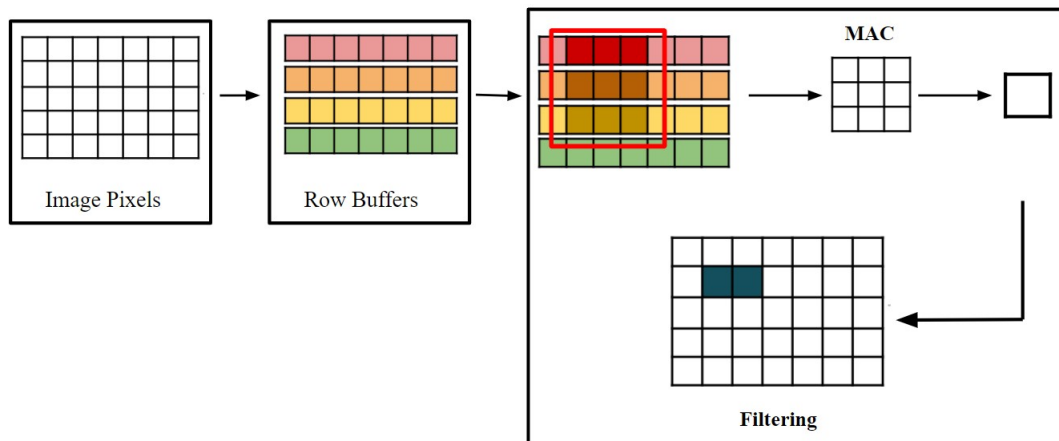


Figure 3.2: Design Flow 2

3.3 Software Specifications

Xilinx Vivado 2020.2 software is used for implementation in Verilog HDL (Hardware Description language).

The image input file is given directly in the software, and the output file is obtained after performing filtering operation in on the input image in .bmp (Bitmap Image file) format.

BMP files are advantageous for storing and showcasing high-resolution digital images since they contain uncompressed data.

Results are obtained for blurring operation and edge detection using a sobel filter on an image of size 512x512. For that, a 3x3 kernel is used.

3.4 Complete Architecture

Figure 3.3 shows the complete architecture of the design, which consists of a Control block that controls the flow of row buffer filled, a MAC unit and a FIFO.

Input to Architecture is s_data which is of 8 bit data that is 1 pixel. Control block's output will have 72 bits data as an input to MAC block. That means 9 pixels as an input to MAC unit will give 1 pixel as an output at a time after filtering.

Row_req_intr input is there to know when the next row of the image is needed in order to fill the row buffers. A detailed explanation is discussed in this section.

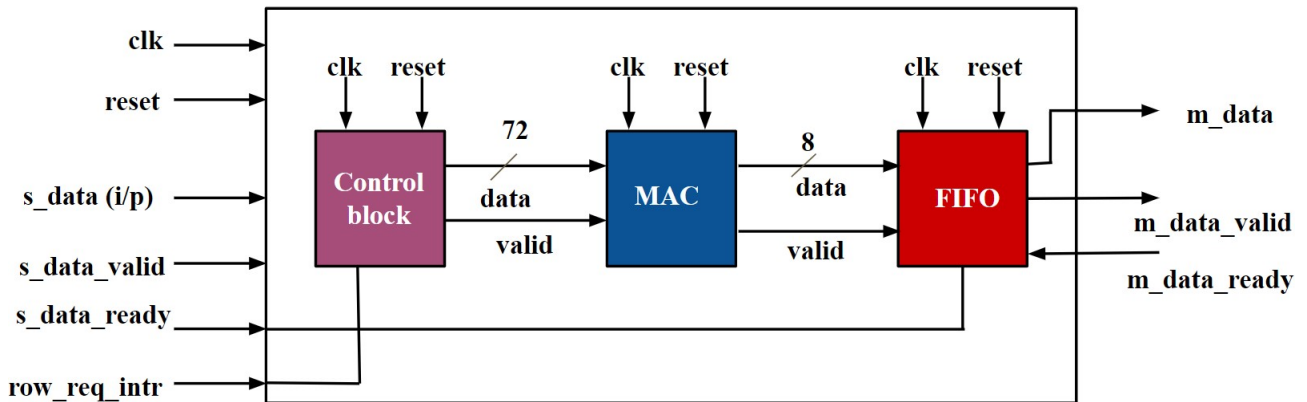


Figure 3.3: Complete Architecture of the Design

3.4.1 Row Buffer

The image used for implementation is 512x512 in size. So for a 512x512 image, one Row buffer will be 512 bytes in size, and three Row buffers will require $512 \times 3 = 1536$ bytes.

The size of the row buffer depends only on the width of the image. Our image is 512x512, so the one-row buffer will be of size 512 bytes, or we can say 512 pixels where each pixel is 8-bit (1 byte = 8 bit).

Basically, IP will wait for three rows of images to produce an output corresponding to one row of the output image and after that, one row of image is needed for every row of the output image.

The same Row buffer can be used multiple times so that row buffers can be reused by overwriting and a multiplexer can be used for processing the data. After the first row is done with the operation, the fourth line buffer will be used. And for that, one extra Row buffer is good to improve the performance else we have to wait till all the row buffers are done. From figure 3.4, we can see that B1, B2, B3 and B4 are the four buffers. While the system is processing three row buffers, data can be sent to fourth buffer so that data transfers happen in parallel to data processing, as shown in figure 3.4.

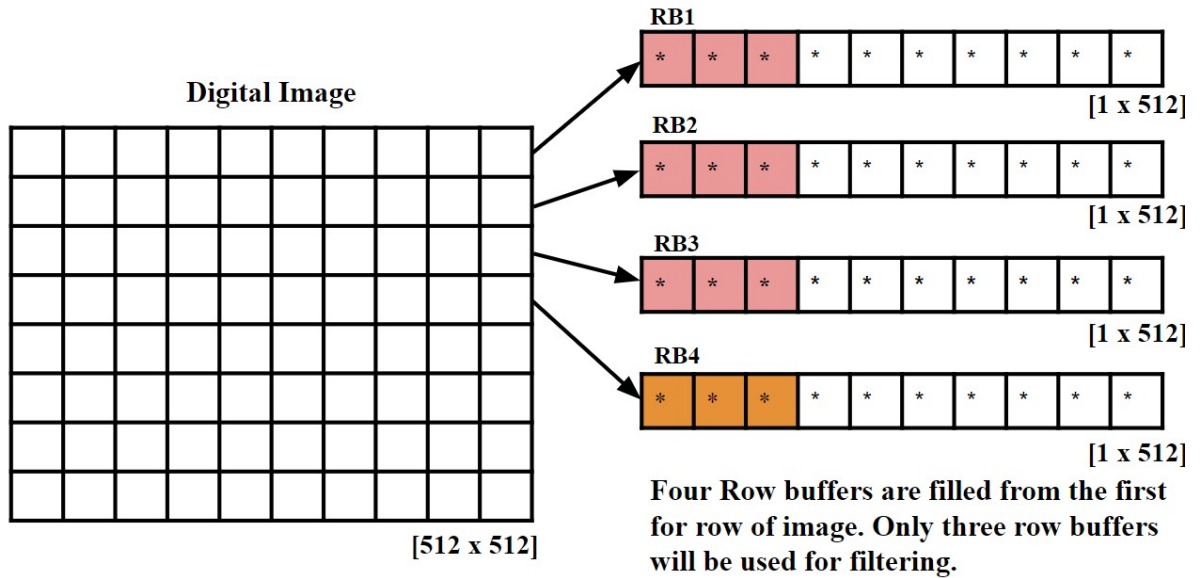


Figure 3.4: Filling of Row buffers

The row buffer IP will take nine input pixels, one pixel at a time, each pixel is of eight bits so the output pixel size of first-row buffer will be twenty-four bits (considering neighbourhood pixels). Similarly, row buffer two and row buffer three will also have 24 bit output respectively. Input and output bits can be shown in figure 3.5.

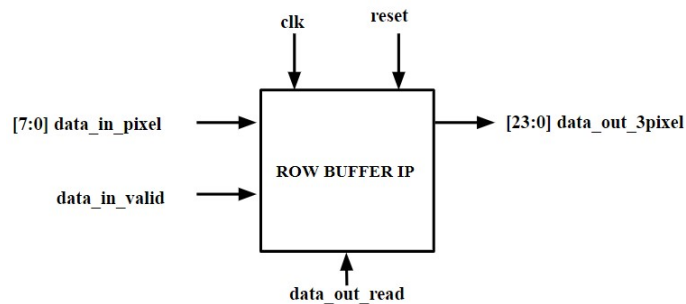


Figure 3.5: Row buffer IP

3.4.2 Control Block

The control block consists of two logics: the write logic and the read logic. The purpose of the write logic is to store data in row buffers, which are instantiated memory units. The write logic block depicted in the figure 3.6 is responsible for generating a valid signal to indicate the appropriate row buffer where the data should be written. It also determines the optimal point to commence writing the data, enabling subsequent filtering operations.

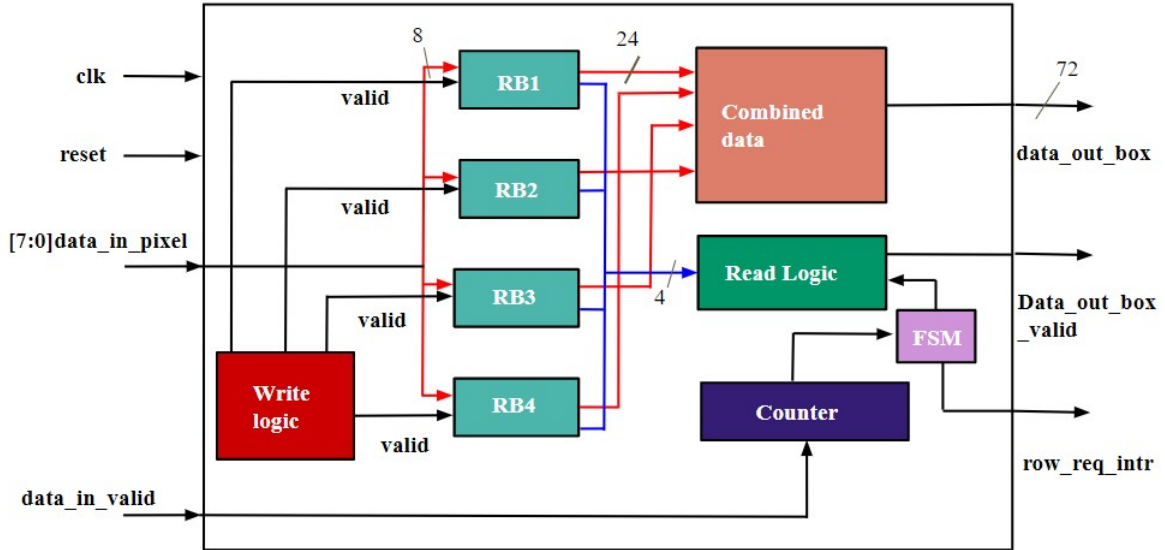


Figure 3.6: Control Block

There are two main tasks associated with the write logic block. Firstly, it needs to determine the specific row buffer where the incoming data should be stored. Secondly, it needs to identify when to initiate the data writing process, allowing for subsequent filtering operations to be performed efficiently.

To perform these tasks, the counter block keeps track of the number of row buffers currently being written. The Finite State Machine (FSM) continuously requests new rows until all the available buffers are filled. Once all the buffers have been utilized, a signal is sent to the FSM, indicating that it should stop requesting new rows.

3.4.3 Multiply and Accumulate (MAC)

Multiply and Accumulate (MAC) operation is as the name suggests will multiply the input image pixels and required kernel to get the multiplied matrix and addition of all the elements of the matrix will give us the output pixel as discussed earlier in chapter 2. For MAC operation, after all the three row buffers are filled,

the size of them will be 3x3 and the size of the kernel used is also 3x3. So MAC block will take 72-bit data as an input and will give 8-bit data as an output that is one pixel, as shown in figure 3.7.

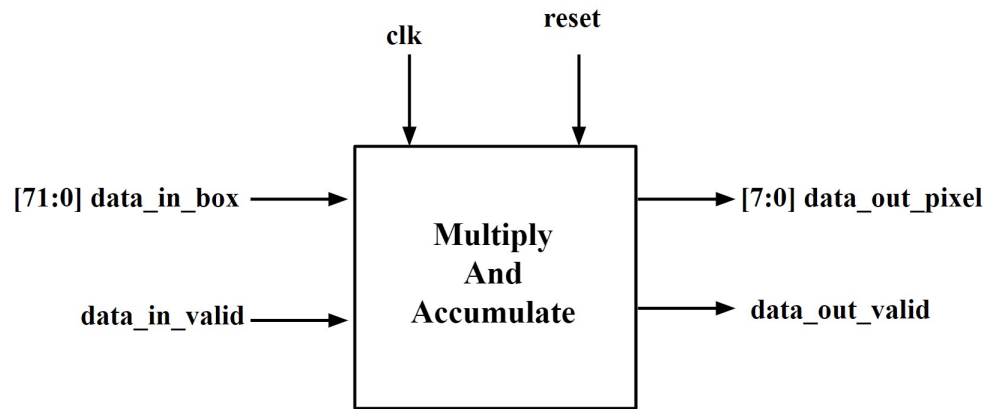


Figure 3.7: Multiply and Accumulate IP

3.4.4 FIFO

A FIFO (First-In, First-Out) block is essential in situations where data is being sent from a source to a destination, but there may be a delay or discrepancy in the reception process. It often takes time for the source to realize that the destination is not receiving the data, and by the time this is identified, there may already be an ongoing process or intellectual property (IP) involved. In order to address this, a FIFO acts as an additional buffer to temporarily store the data until it can be successfully transmitted to the destination.

3.5 Simulation Results

Various simulation results are obtained after implementing the previously explained design. Results include RTL schematic of the design, Output Waveform, Output Images and Resource Utilization.

3.5.1 RTL Schematic

RTL schematic of the design is shown in the figure 3.8. It can be seen that there are three blocks that we discussed: c1(control block), m1(Multiply and Accumulate, and f1(FIFO).

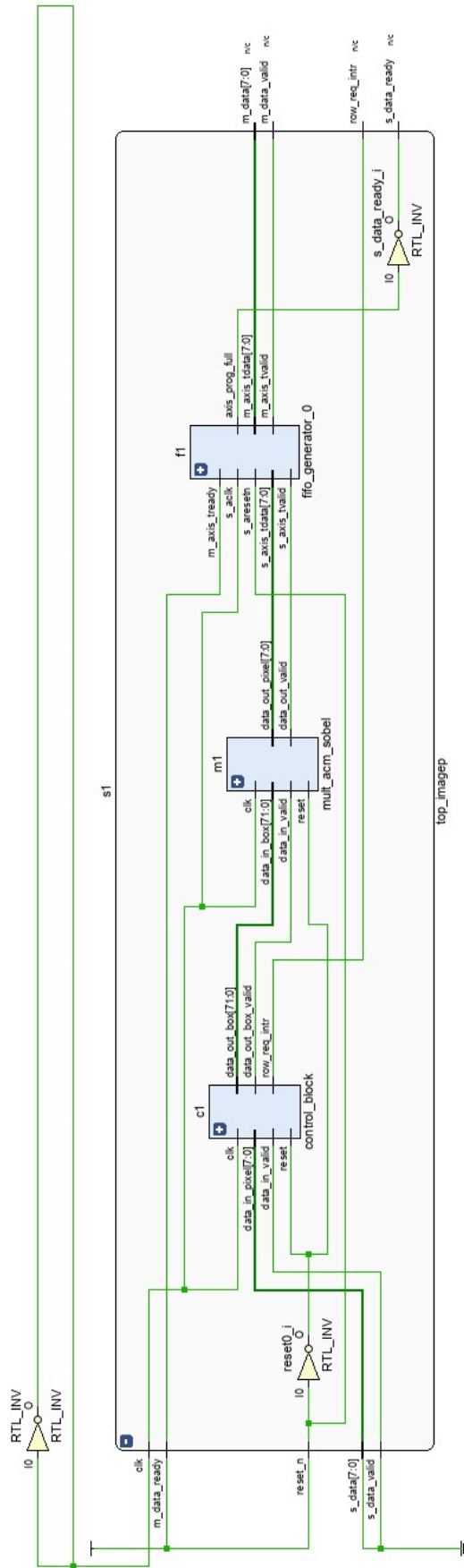


Figure 3.8: RTL Schematic of the design

3.5.2 Output Images

Time taken to process the image was 2.6 ms. Figure 3.9 and 3.10 shows the output images for the implemented design. Two operations were performed: Blurring of an image and Sobel edge detection of an image. An image was first converted to gray scale image and then both the filtering operation were carried out. So in figures 3.9 and 3.10, First Image is the gray scale image, the second image is the blurred image obtained from the first image and the third image in the horizontal sequence is the sobel edge detection of an image as sobel edge detection technique that plays a crucial role in various image processing and computer vision applications, including object recognition, segmentation, tracking, and enhancement.

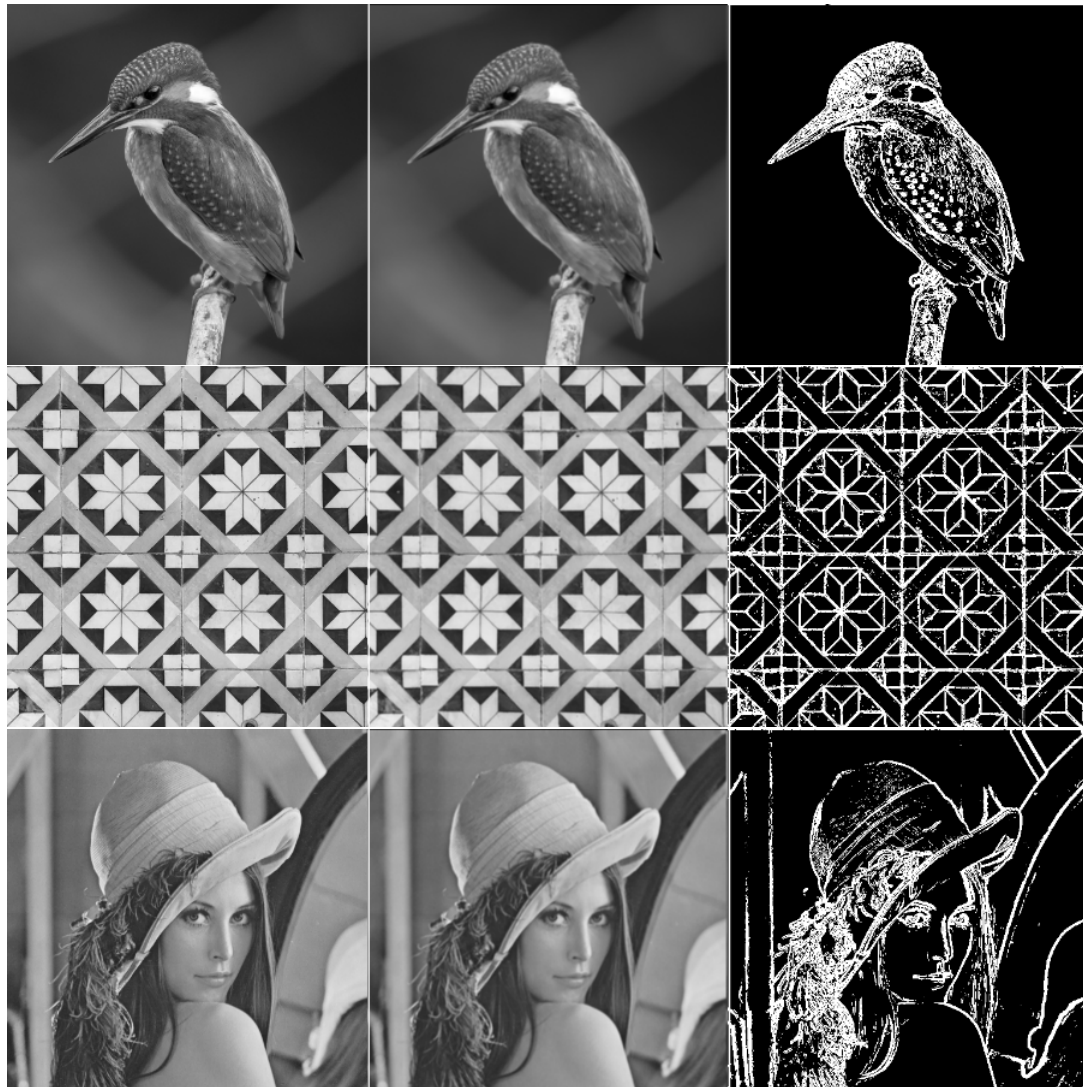


Figure 3.9: (1) Gray Image (2) Blur filtering (3) Sobel Edge detection

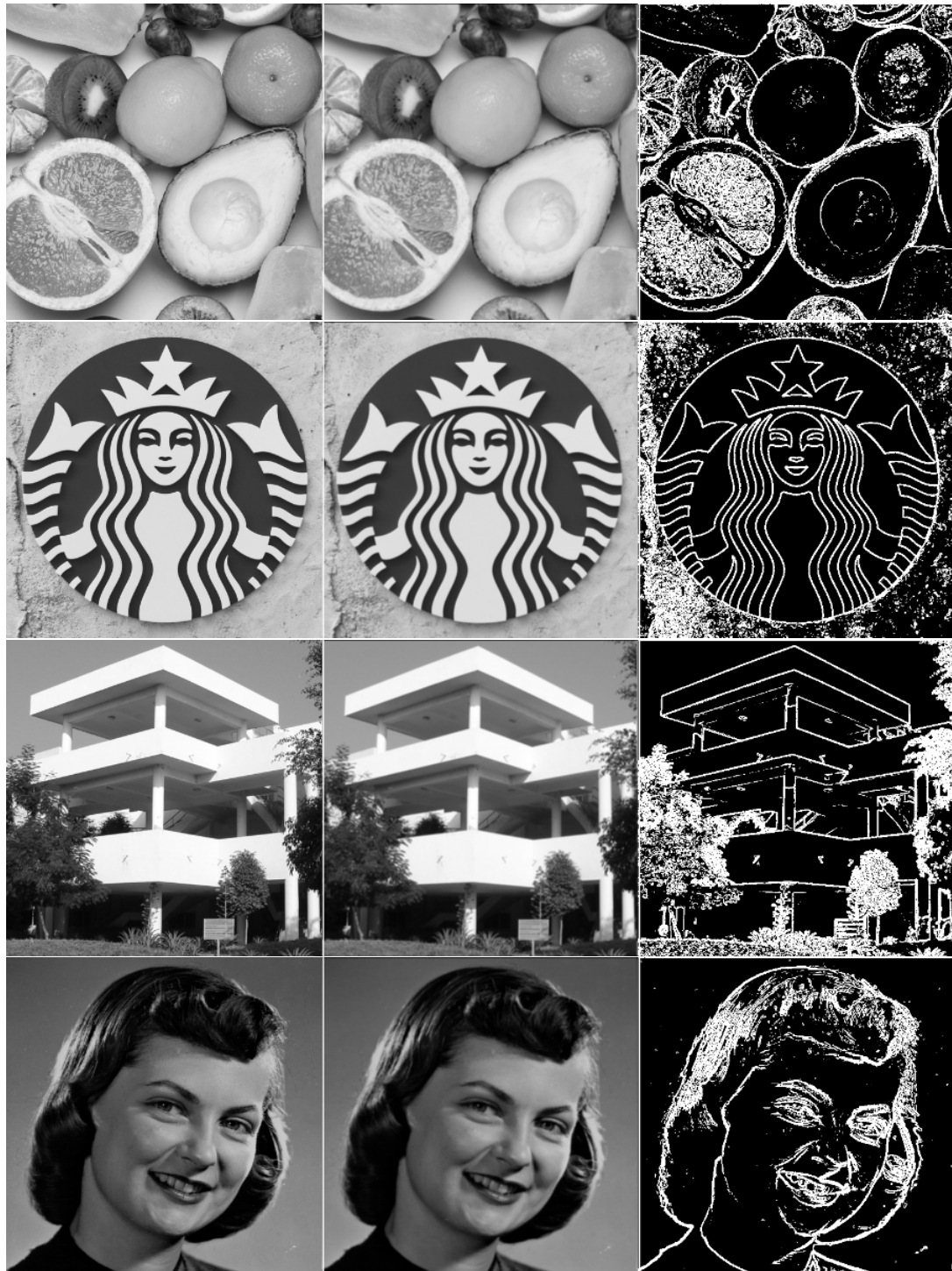


Figure 3.10: (1) Gray Image (2) Blur filtering (3) Sobel edge detection

3.5.3 Output Waveform

The output waveform of the implemented design is shown in Figure 3.11. Waveform consists of the image pixels both input and output as mentioned in figure 3.11. Each new pixel is fetched at the positive edge of the clock and also the address is incremented at the positive edge of the clock. Pixels sent and received information can also be seen.

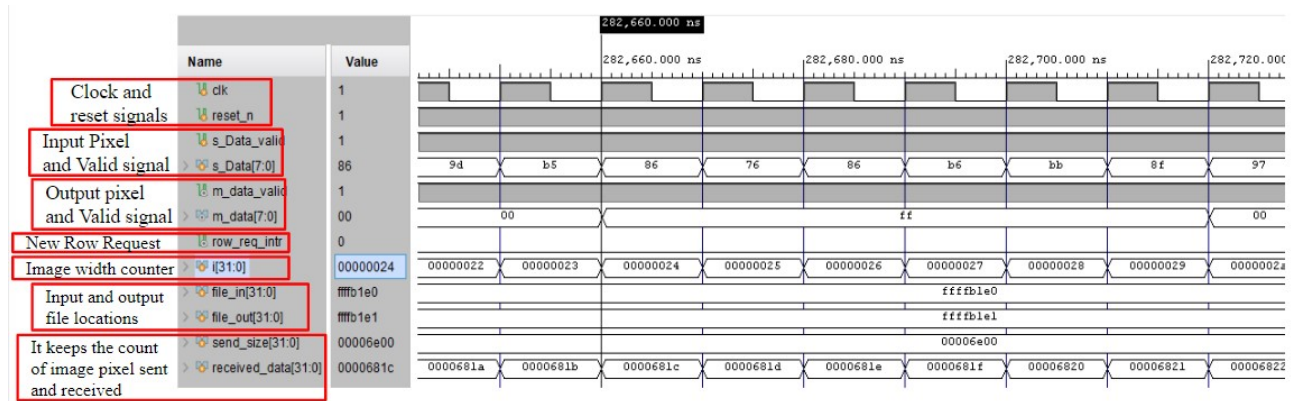


Figure 3.11: Output Waveform

3.5.4 Utilized Resources

From the complete Architecture, we saw there were three blocks: Control block, MAC unit and FIFO, Figure 3.12 shows the devices used for the whole design and individually by each block. Figure 3.13, Utilization of resources based on the available resources are shown.

Name	^1	Slice LUTs (10400)	Slice Registers (20800)	Block RAM Tile (25)	DSPs (45)	BUFGCTRL (32)
image_proc_tb		264	182	0.5	2	1
s1 (top_imagep)		263	182	0.5	2	0
c1 (control_block)		65	64	0	0	0
f1 (fifo_generator_0)		57	61	0.5	0	0
U0 (fifo_generator_0)		57	61	0.5	0	0
m1 (mult_acm_sobel)		141	57	0	2	0

Figure 3.12: Hierarchy of the design

Figure 3.14 shows the power consumption. 96% dynamic power is consumed out of which 48% is consumed by the signals, 46% is consumed by the logic and the remaining is consumed by the BRAM. While 4% static power is consumed.

Resource	Utilization	Available	Utilization %
LUT	264	10400	2.54
LUTRAM	2	9600	0.02
FF	182	20800	0.88
BRAM	0.50	25	2.00
DSP	2	45	4.44

Figure 3.13: Resource Utilization

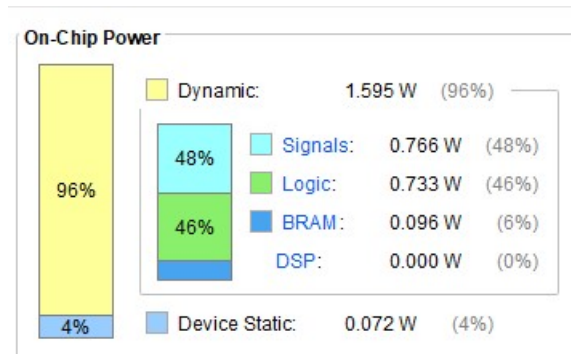


Figure 3.14: Power consumption

3.6 Conclusions

In this Chapter, Sobel edge detection and blurring filter were applied to images with a size of 512x512 pixels. However, the approach discussed can be adapted to work with images of any size as long as the FPGA (Field-Programmable Gate Array) has enough memory to hold the image.

The statement mentions the use of four-row buffers during image processing. These row buffers are temporary storage areas that hold a portion of the image data. By using four-row buffers, the processing time was reduced compared to using only three-row buffers. This is because having an extra buffer allows for a continuous flow of data, and the processing can continue without waiting for the fourth buffer to be filled.

Time taken to process the image using all four buffers was 2.6 milliseconds. Now, if five buffers were used instead of four, the time taken to process the image would be even lesser. The exact reduction in processing time cannot be determined without additional information or performance measurements. However, the addition of an extra buffer provides more flexibility in data transfer and processing, which generally results in improved efficiency and reduced processing time.

CHAPTER 4

Proposed Method

A method is proposed in this thesis that takes the input image and performs linear filtering on that image and gives us the final output image which is displayed through VGA on the monitor. The block diagram of the proposed method is shown in figure 4.1.

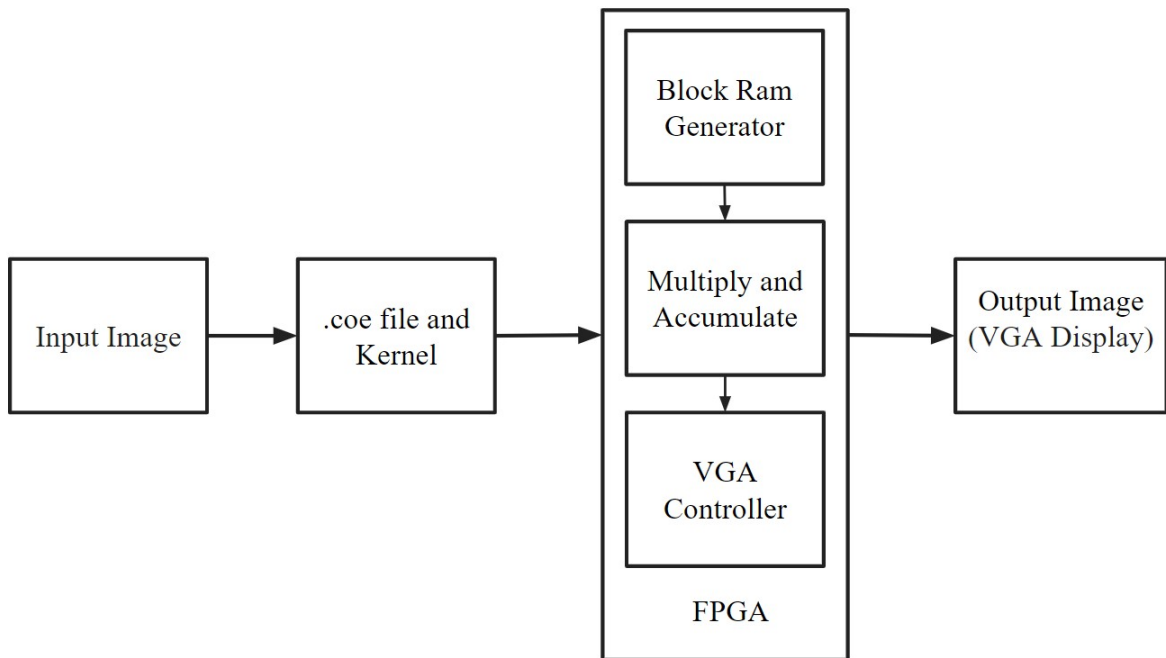


Figure 4.1: The Block diagram of proposed method

4.1 COE file

A COE file is a text file consisting of a radix header and multiple vectors, where each vector ends with a semicolon. The radix in the header can have a value of 2, 10, or 16. A COE file is then translated into an ASCII text file called Memory Initialization File (.mif file). This COE file is generally used for Block memories.

A COE file for an image is a plain text file that contains the pixel values in a specific order, often in binary format. Information like the number of rows and columns in the image and the number of bits used to represent each pixel are included in the form of a COE file for an image.

COE files for images are often used in hardware implementations of image processing algorithms. By storing the image data in a COE file, the image can be loaded directly into the hardware without requiring an external memory interface. This can be particularly useful in applications where real-time image processing is required, such as in video processing, image processing, and machine vision applications.

4.2 Block RAM Generator

The Xilinx IP Block Memory Generator (BMG) is a tool for constructing memories that takes advantage of the embedded block RAM resources available in Xilinx FPGAs. This tool generates memories that are optimized for both area and performance, allowing users to easily create customized memories that fully utilize the capabilities of Xilinx FPGAs.

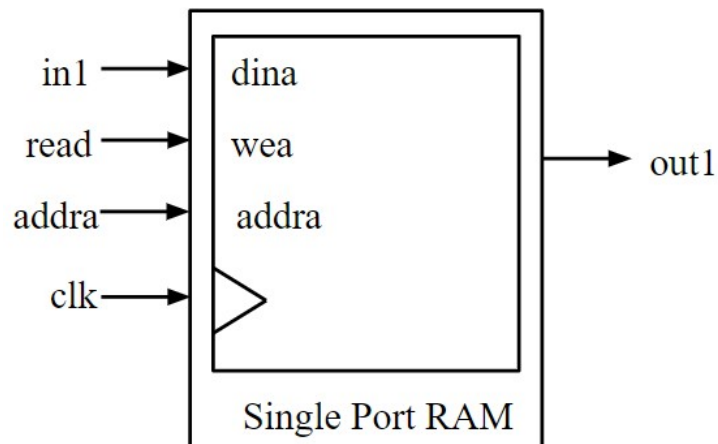


Figure 4.2: Block RAM

The IP Block Memory Generator simplifies the process of creating memory structures within an FPGA design. It allows designers to generate memory cores with various configurations, such as size, data width, and type of memory (RAM, ROM, FIFO, etc.). The generated IP cores can be integrated into the overall FPGA design to implement on-chip memory requirements.

COE file is given in the IP Block RAM Generator where image width and depth

has to mentioned for the process. Single port RAM is selected in this method which can be seen in figure 4.2.

4.3 Multiply and Accumulate

This is the part where filtering of the input image is performed as per the steps mentioned in previous chapter under filtering section. For example let us consider an input image A in equation 4.1 and filter B in equation 4.2.

$$\text{Image A} = \begin{bmatrix} 44 & 34 & 07 & 23 \\ 80 & 88 & 1 & 65 \\ 134 & 12 & 66 & 111 \\ 13 & 125 & 145 & 11 \end{bmatrix} \quad (4.1)$$

$$\text{Kernel B} = \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} \quad (4.2)$$

Now, If we want to work on pixel 88 in image A then we will position a Kernel or filter over each pixel in the image matrix and multiply. This multiplication is carried out for each element of the filter as seen in the equation 4.3.

$$\text{multiply} = \begin{bmatrix} 44 & 34 & 07 \\ 80 & 88 & 1 \\ 134 & 12 & 66 \end{bmatrix} * \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -44 & 0 & -07 \\ 0 & 325 & 0 \\ -134 & 0 & -66 \end{bmatrix} \quad (4.3)$$

The products obtained from the multiplication are added together and one output pixel is obtained as seen in equation 4.4

$$\text{Output pixel} = -44 + 0 - 07 + 0 + 352 + 0 - 134 + 0 - 66 = 101 \quad (4.4)$$

So, the final pixel we get is 101. The same procedure is repeated for all the image pixels and after processing every pixel we will get the final filtered output image. When processing grayscale images, a common issue arises where the output of the layers becomes smaller compared to the input. This occurs because the output size is determined by the input size and the filter/kernel used. Specifically, for an input image of size $(n \times n)$ and a filter/kernel of size $(k \times k)$, the output size is $(n - k + 1) \times (n - k + 1)$. This reduction in output size can lead to the loss of valuable

information, particularly at the corners of the image. Since the filters move across the pixels, they may not fully focus on the corners during this process. As a result, the reduced information in the corners can cause confusion for subsequent layers in the network.

4.3.1 Padding

To solve the problem discussed earlier, padding layers can be employed. Padding involves adding additional pixels around the borders of the input image before applying the filters. This allows the filters to cover the corner pixels and reduces the loss of information. By padding the input image, the output size of the layers can be preserved or adjusted to match the input size, mitigating the information loss and confusion in subsequent layers. Let us understand using the above example. Padding zeros in image A as seen in equation 4.5 and using filter B in equation 4.6.

$$\text{Image A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 44 & 34 & 07 & 23 & 0 \\ 0 & 80 & 88 & 1 & 65 & 0 \\ 0 & 134 & 12 & 66 & 111 & 0 \\ 0 & 13 & 125 & 145 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.5)$$

$$\text{Kernel B} = \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} \quad (4.6)$$

Now, If we want to work on pixel 44 in image A, we will position a Kernel or filter over each pixel in the image matrix and multiply. This multiplication is carried out for each element of the filter as seen in the equation 4.7.

$$\text{multiply} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 44 & 34 \\ 0 & 80 & 88 \end{bmatrix} * \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 176 & 0 \\ 0 & 0 & -88 \end{bmatrix} \quad (4.7)$$

The products obtained from the multiplication are added together and one output pixel is obtained as seen in equation 4.8

$$\text{Output pixel} = 0 + 0 + 0 + 0 + 0 + 176 + 0 + 0 + 0 - 88 = 88 \quad (4.8)$$

So, the final pixel we get is 88. Thus, the problem of output size reduction and loss of information in the corners of grayscale images can be alleviated by incorporating padding layers. Padding helps maintain the size of the output and ensures that important information from all parts of the image is considered during the filtering process.

4.4 VGA Controller

After getting our filtered output image we have to display it on monitor to visualize the output image. VGA controller is used to display the image on the monitor.

Video Graphics Array is a type of display technology that was introduced in the late 1980s. It is a type of analog video standard that is widely used in personal computers and laptops. A VGA display has a resolution of 640x480 pixels as shown in Figure 4.3, it can be seen that active part is of size 640x480 and rest of the pixels are used for synchronization to display the image and uses a 15-pin connector to connect the computer to the display. It can display up to 16 colours at once or up to 256 colours in a lower-resolution mode [25]. The front porch is a blanking interval that occurs before the horizontal sync pulse. It is a period during which the electron beam in a CRT monitor is turned off and allowed to move back to the left side of the screen to start drawing the next line.

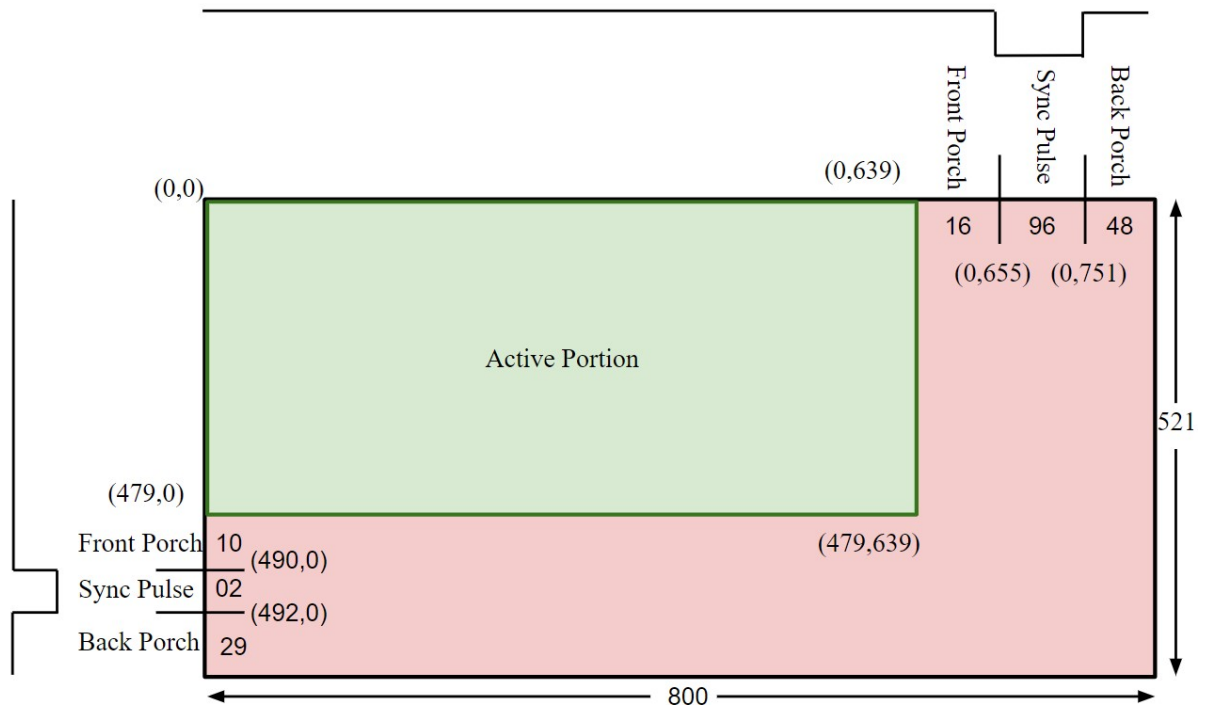


Figure 4.3: VGA Display

The front porch is used to ensure that the electron beam has enough time to move back to the left side of the screen before the start of the next line.

The back porch is a blanking interval that occurs after the horizontal sync pulse. It is a period during which the electron beam is turned off again and allowed to move to the right side of the screen to start drawing the next line. The back porch is used to ensure that the electron beam has enough time to move to the right side of the screen before the start of the next line.

The horizontal sync pulse is a short pulse that occurs during the horizontal blanking interval. It is used to synchronize the display with the incoming video signal. The horizontal sync pulse tells the display when to start drawing the next line.

The vertical sync pulse is a short pulse that occurs during the vertical blanking interval. It is used to synchronize the display with the incoming video signal. The vertical sync pulse tells the display when to start drawing the next frame.

Thus, horizontal sync and vertical sync timing signals are produced by a VGA controller circuit, which is also in charge of synchronising the supply of visual data with the pixel clock.

The pixel clock specifies the duration for showing a single data pixel. The VS signal sets the display's refresh frequency or the rate at which all of the data on the screen is regenerated. Based on observations, the timings for the sync pulse width and the front and rear porch intervals (the moments before and after the sync pulse when no information may be displayed) were determined [25].

The signal timings for a 640x480 monitor with a refresh rate of 60 +/-1Hz and a 25 MHz pixel clock can be derived according to the standard timings [25].

CHAPTER 5

Implementation of the Proposed method

Implementation of the proposed method is needed to validate and verify its effectiveness in the hardware environment. All the filtering operations are carried out on an image size of 200x200 pixels.

5.1 Hardware and Software Used

5.1.1 Hardware Required

Nexyx 4 DDR Artix-7 FPGA board is used. The Nexys 4 DDR board is a fully-equipped development platform for digital circuit design. It features the latest Artix-7 Field Programmable Gate Array (FPGA) from Xilinx, offering a large and high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C). The board includes ample external memory and various ports such as USB, Ethernet, and more, making it suitable for a very wide range of projects from basic combinational circuits to advanced embedded processors. With built-in peripherals like an accelerometer, temperature sensor, MEMs digital microphone, speaker amplifier, and multiple I/O devices, the Nexys4 DDR can be used for diverse designs without requiring additional components. VGA Male to Female Connector cable and a monitor to display the image is also required.

5.1.2 Software Required

Xilinx Vivado 2020.2 Software, Vivado provides a comprehensive set of tools and features for FPGA and SoC development, including Register Transfer Level (RTL) synthesis, high-level synthesis (HLS), Intellectual Property (IP) integration, system-level integration, and debugging capabilities. It also supports various design entry methods like VHDL, Verilog, and SystemVerilog.

Google Colab is a cloud-based platform provided by Google that allows users to run Python code in a Jupyter Notebook environment. It was used to generate COE file.

Language: Verilog HDL (Hardware description Language) and Python

5.2 Complete Architecture of Proposed Method

Complete architecture of the proposed method is shown in the figure 5.1. It consists of various blocks such as:

VGA controller which will synchronize the horizontal sync pulse and vertical sync pulse so that the image is displayed perfectly on the monitor screen via VGA cable without any difficulties.

Block RAM IP is used to take input images in the form of binary pixels from the text COE file, which contains details of the width and depth of the image. The COE file needs to be uploaded during the generation of the IP. Block RAM also has an address counter that keeps the count of addresses at which our pixel processing is being carried out. It will continue to check whether all the pixels are processed or not, if not then it will increment the counter else it will end the counter. Here image size is 200x200 so we have in total 400,000 pixels therefore address counter will count to 39,999 and we will get the filtered output image.

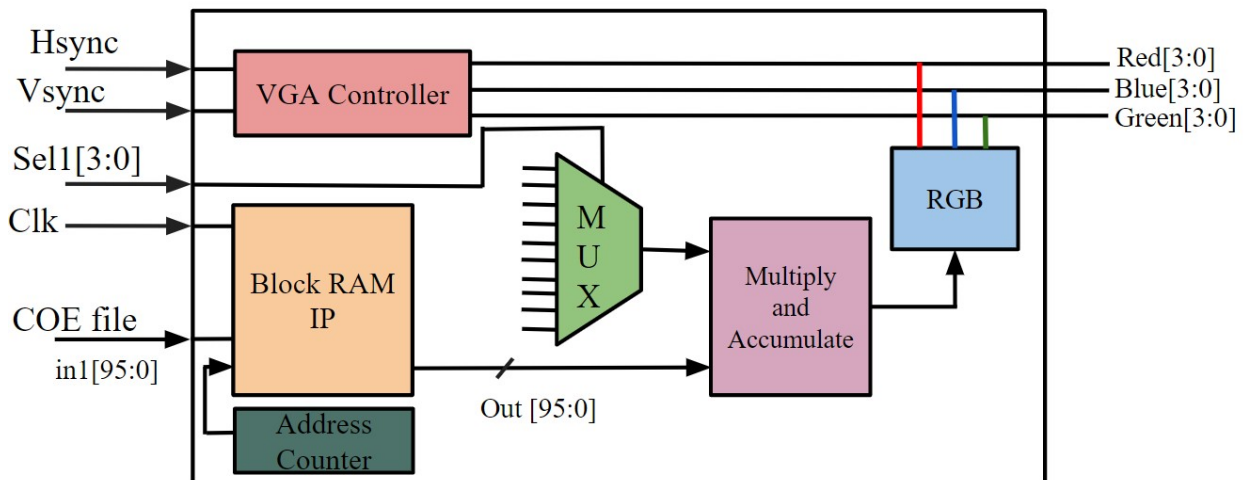


Figure 5.1: Complete Architecture

Multiplexer is used here to select the filter that needs to be implemented. A total of ten images can be displayed based on the selection line of the multiplexer. We have used a 16x1 multiplexer with 16 selection lines and that has been later customized with our requirements and the number of filters.

After we get our input image pixels through COE file and filter that need to be implemented as the output of the multiplexer, Multiply and accumulate procedure starts as discussed in chapter 3.

Once the Multiply and Accumulate step is done, the output is displayed directly on the monitor via VGA cable.

5.3 COE file generation

Using Python language and Google Colab software COE file is generated, it consists of vectors of image pixels in binary format. The input image is RGB image

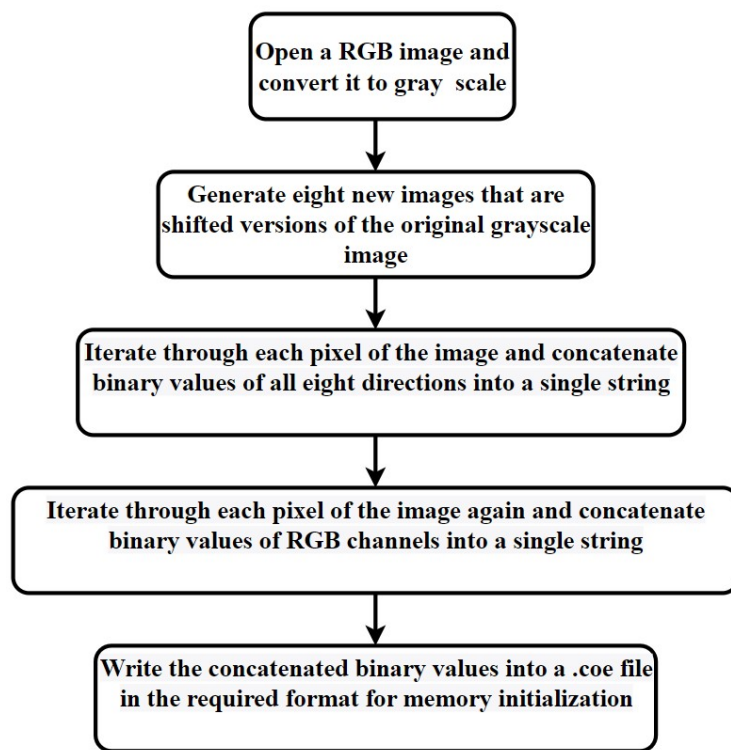


Figure 5.2: COE file Generation

and it can be in any format such as .jpg,.png,.bmp, etc. So, first, the image is converted into gray scale image because for filtering operations, images need to be converted into gray scale which was studied earlier in Chapter 2. Eight new images are generated which are shifted versions of the gray scale image such as left shifted, right shifted, shifted upwards, shifted downwards, left-up shifted, left-down shifted, right-up shifted and right-down shifted because we need to multiply this with the filter to get the output pixels. Now, this is then converted to binary form and is concatenated in a COE file with Width for one vector being 72-bit. After that to display the original image we also need the RGB pixel values and

5.5 Block RAM Specifications

The size of block RAM for Nexys 4 DDR is 4,860 Kb. Block Memory Generator can generate memory structures from 1 to 4096 bits wide and from 2 to 1048576 locations deep based on the size of BRAM. So width x depth needs to be such that the size should not exceed the actual size of BRAM.

5.6 VGA Specifications

The Nexys 4 DDR board has 14 FPGA signals to create a VGA port with 4 bits-per-color Red, Green and Blue. It has two pins for standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync). 4096 different colors can be displayed, one for each unique 12-bit pattern. 12 bits are interfaced with the 12ins of VGA cable. Figure 5.4 shows the VGA connector and FPGA interfacing pins [25].

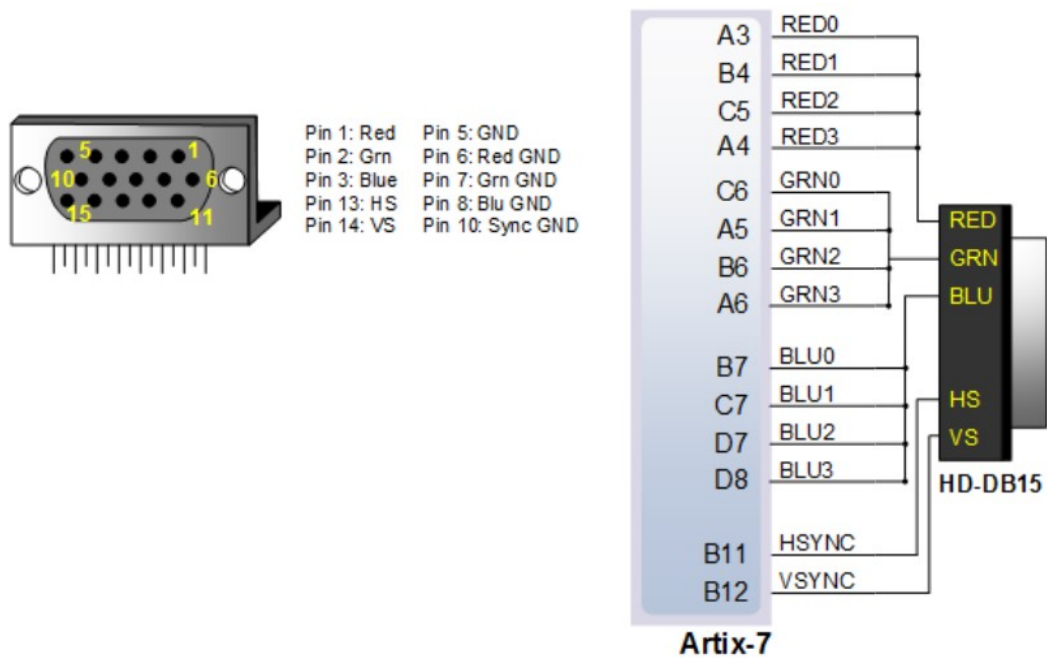


Figure 5.4: VGA interfacing with FPGA

5.7 Flow of Proposed Method

The following Figures 5.5, 5.6 and 5.7 show the flow of filtering and displaying the image on the monitor.

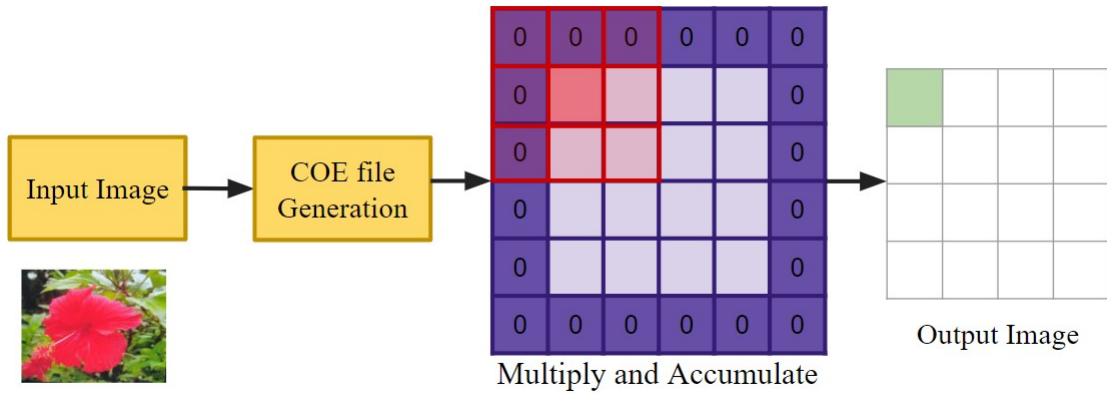


Figure 5.5: Flow of Proposed Method (1): Filtering on 1st pixel

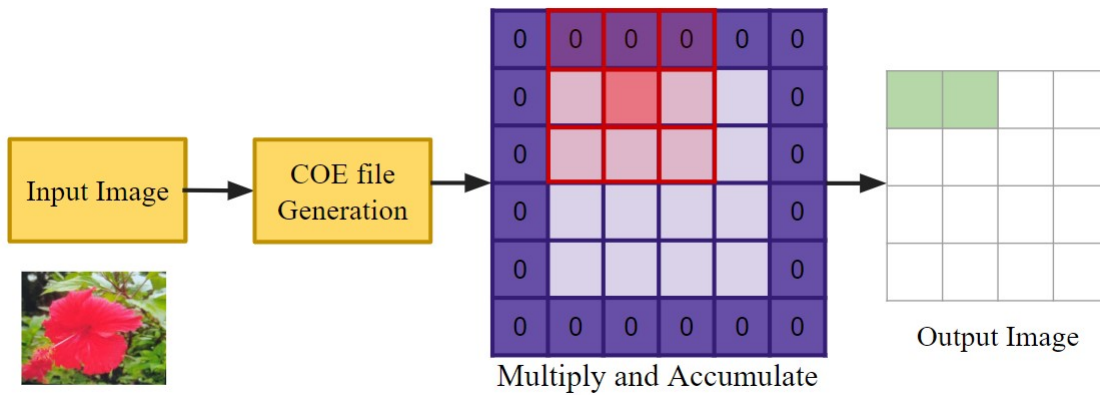


Figure 5.6: Flow of Proposed Method (2): Filtering on 2nd pixel

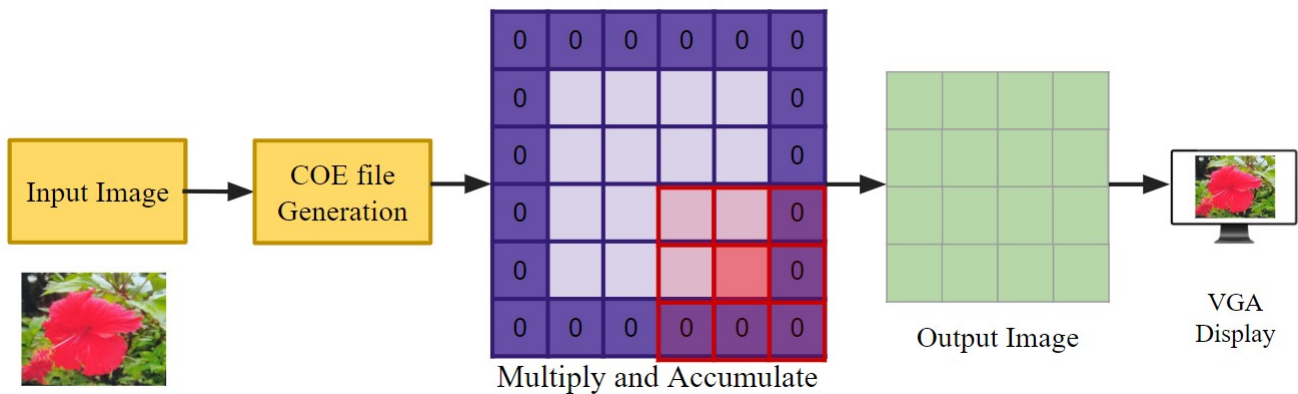


Figure 5.7: Flow of Proposed Method (3): Output Image displayed on monitor after filtering process was completed

5.8 Hardware Setup

Figure 5.8 shows the hardware setup of the proposed method. The FPGA board is connected to Laptop through USB cable through which code is dumped on FPGA and another connection through FPGA is to VGA port as seen in Figures 5.8 and 5.9.

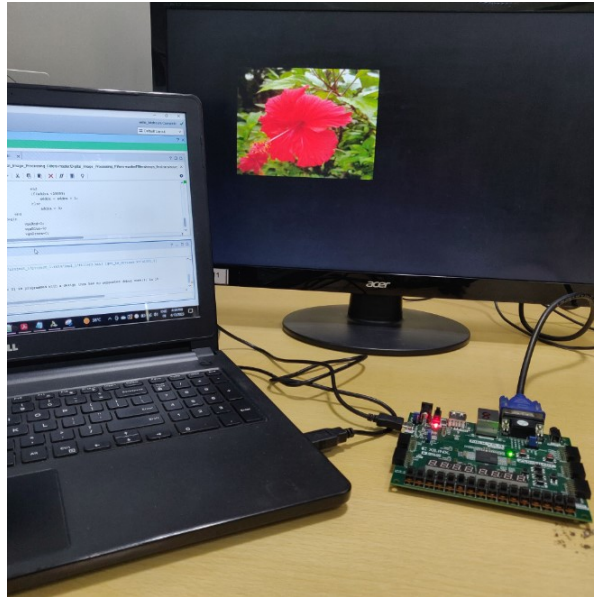


Figure 5.8: Hardware Setup

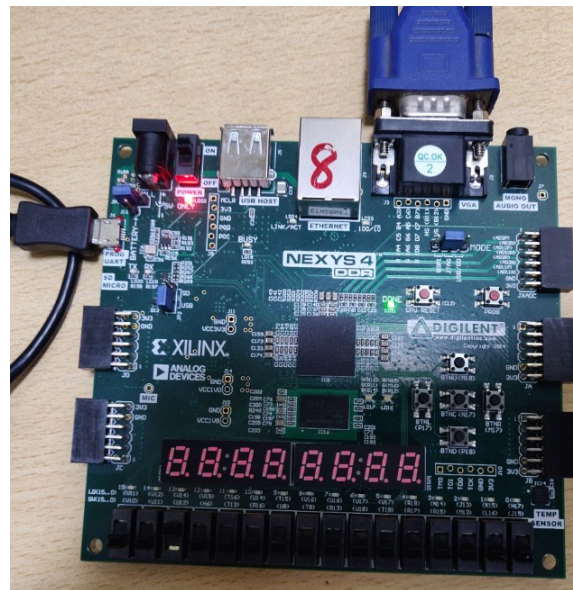


Figure 5.9: FPGA board

CHAPTER 6

Simulation and Hardware Results

Simulation and Hardware results are obtained for the proposed method. Simulation results contain the output waveform generated by the final output image, resources used and power consumption. Hardware results are the output image in a visual form which is observed on the monitor.

6.1 Output Waveform of the Proposed Method

The waveform is the digital data of the output image. As studied earlier, digital image is an image which contains pixel values. It can be seen from the output waveform 6.1 that pixels are changing as per the clock and operations are being performed based on the filter selected.

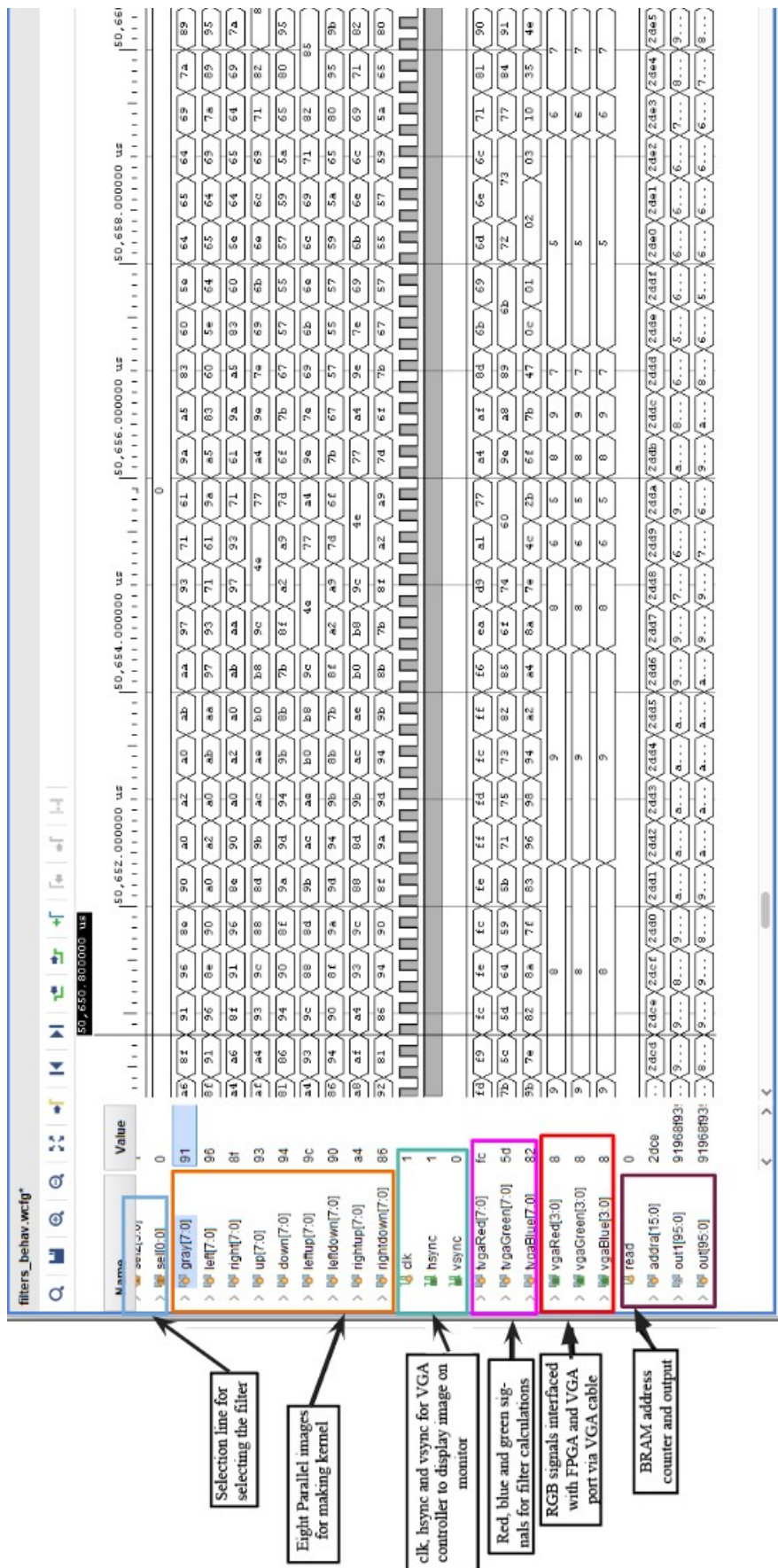


Figure 6.1: Output Waveform

6.2 FPGA Implementation Results

Results obtained from FPGA implementation and output images were observed on a monitor with resolution of 640x480. From figure 6.2, it can be seen that nine filters are used on ten different images to see the results. In figure 6.2, the first image (1) is the original RGB image displayed on the monitor, image (2) is the grey image converted from the original RGB image, image (3) is the sobel filter applied on the gray image (2) we can see that it performs edge detection of the gray image.



Figure 6.2: (1) Original (2)Gray (3)sobel (4)SobelX (5)SobelY (6)Gaussian blur (7)Prewitt X (8)Prewitt Y (9)Laplacian 8 (10)Laplacian 4

Image (4) is sobel X filter, and it can be seen that the edges that are detected are parallel to the y-axis because this filter is such that it transitions from dark to

light on the right side and from light to dark on the left side.

Image (5) is sobel Y filter, and it can be seen that the edges that are detected are parallel to the x-axis because this filter is such that it transitions from dark to light at the bottom and from light to dark at the top.

Image (6) is the Gaussian blur filter which blurs the filter and helps to remove the noise in image enhancement process.

Image (7) and image (8) are Prewitt edge detection filter, and it works almost similar to the sobel X and sobel Y edge detection filters respectively. But the difference is that the sobel filter emphasizes more on the nearby neighbourhood pixels by increasing the weight of absolute nearby pixels. In contrast, in Prewitt filter emphasizes each neighbourhood pixels equally.

Image (9) and image (10) are laplacian edge detection filters that are symmetrical from all sides.

6.3 Utilized FPGA Hardware Resources

Below shown reports are the combined Utilization report of the method. As seen in Figure 6.3. The Block RAM tile available is 135, and 106.5 is used out of that, which is the maximum out of all the resources. Power consumption was maximum due to BRAM as shown in Figure 6.4.

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Block RAM Tile (135)	Bonded IOB (210)	BUFGCTRL (32)
filters	1114	156	90	106.5	21	1
inst1 (blk_mem_gen_0)	365	14	90	106.5	0	0
U0 (blk_mem_gen_0)	365	14	90	106.5	0	0

Figure 6.3: Resource Utilization

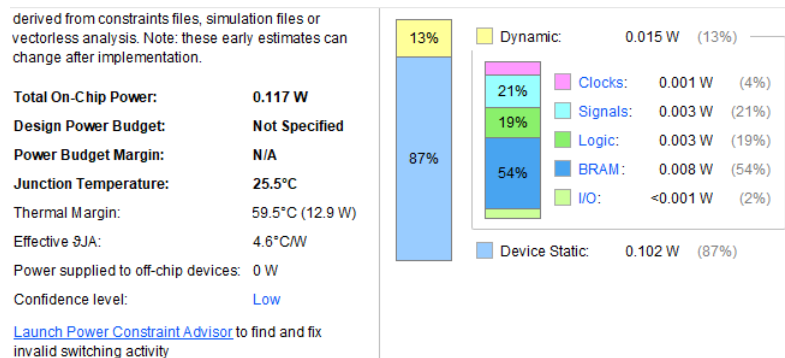


Figure 6.4: Power Consumption

Total dynamic power is 0.105, and static power is 0.102. All the resources that were utilized during this implementation is shown figure 6.5.

Resource	Utilization	Available	Utilization %
LUT	1114	63400	1.76
FF	156	126800	0.12
BRAM	106.50	135	78.89
IO	21	210	10.00

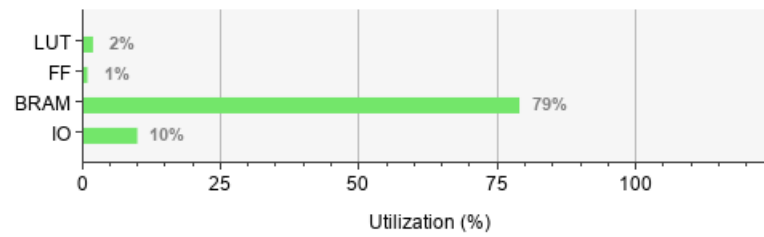


Figure 6.5: Utilization summary

CHAPTER 7

Conclusions

Digital programming techniques were utilized to apply different filters in image processing algorithms, which were then displayed on a monitor through VGA interfacing. The experiments were conducted using the Nexys 4 DDR Artix-7 FPGA board with an image size of 200x200.

In conclusion, this research demonstrates the potential of digital programming and FPGA technology in implementing image processing algorithms. The results suggest that it is possible to achieve effective image filtering and display outputs through VGA interfacing.

These findings may have significant implications for the development of real-time image processing applications in various fields such as medicine, security, and robotics. Further research could explore the feasibility of scaling up the image size and complexity to achieve even more advanced image processing capabilities.

References

- [1] T. S. Huang, W. F. Schreiber, and O. J. Tretiak, "Image processing," *Proceedings of the IEEE*, vol. 59, no. 11, pp. 1586–1609, 1971.
- [2] Z. I. Azhari, S. Setumin, A. D. Rosli, and S. J. A. Bakar, "A systematic literature review on hardware implementation of image processing," *International Journal of Reconfigurable and Embedded Systems*, vol. 12, no. 1, p. 19, 2023.
- [3] F. Merchant and K. Castleman, *Microscope image processing*. Academic press, 2022.
- [4] E. Onat, "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)*. IEEE, 2017.
- [5] D. G. Bailey, "Image processing using fpgas," p. 53, 2019.
- [6] S. S. Antora, Y. K. Chang, T. Nguyen-Quang, and B. Heung, "Development and assessment of a field-programmable gate array (fpga)-based image processing (fip) system for agricultural field monitoring applications," *AgriEngineering*, vol. 5, no. 2, pp. 886–904, 2023.
- [7] K. Pauwels, M. Tomasi, J. Diaz Alonso, E. Ros, and M. M. Van Hulle, "A comparison of fpga and gpu for real-time phase-based optical flow, stereo, and local image features," *IEEE Transactions on Computers*, vol. 61, no. 7, pp. 999–1012, 2012.
- [8] M. Isik, K. Inadagbo, and H. Aktas, "Design optimization for high-performance computing using fpga," *arXiv preprint arXiv:2304.12474*, 2023.
- [9] "Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms," *Signal Processing: Image Communication*, vol. 68, pp. 101–119, 2018.

- [10] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of fpga, gpu and cpu in image processing," in *2009 International Conference on Field Programmable Logic and Applications*, 2009.
- [11] W. Gao and Q. Kema, "Parallel computing in experimental mechanics and optical measurement: a review," *Optics and Lasers in Engineering*, vol. 50, no. 4, pp. 608–617, 2012.
- [12] "Medical image processing on the gpu – past, present and future," *Medical Image Analysis*, vol. 17, no. 8, pp. 1073–1094, 2013.
- [13] W. Shi, F. Ji, L. Lan, S.-C. Liang, H.-N. Ding, H. Wang, N. Li, Q. Li, X.-Q. Li, and Q.-J. Wang, "Characteristics of cochlear microphonics in infants and young children with auditory neuropathy," *Acta Oto-Laryngologica*, vol. 132, no. 2, pp. 188–196, 2012.
- [14] O. Fluck, C. Vetter, W. Wein, A. Kamen, B. Preim, and R. Westermann, "A survey of medical image registration on graphics hardware," *Computer methods and programs in biomedicine*, vol. 104, no. 3, pp. e45–e57, 2011.
- [15] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, "A survey of medical image registration on multicore and the gpu," *IEEE signal processing magazine*, vol. 27, no. 2, pp. 50–60, 2010.
- [16] M. M. Petrou and C. Petrou, *Image processing: the fundamentals*. John Wiley & Sons, 2010.
- [17] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*. John Wiley & Sons, 2011.
- [18] L. Busin, N. Vandenbroucke, and L. Macaire, "Color spaces and image segmentation," *Advances in imaging and electron physics*, vol. 151, no. 1, p. 1, 2008.
- [19] L. Shuhua and G. Gaizhi, "The application of improved hsv color space model in image processing," in *2010 2nd International Conference on Future Computer and Communication*, vol. 2, 2010.
- [20] N. John, A. Viswanath, V. Sowmya, and K. Soman, "Analysis of various color space models on effective single image super resolution," in *Intelligent Systems Technologies and Applications: Volume 1*. Springer, 2016.
- [21] S. S. Andrews, "Color," in *Light and Waves: A Conceptual Exploration of Physics*. Springer, 2023.

- [22] E. Olmedo, J. De La Calleja, A. Benitez, and M. A. Medina, "Point to point processing of digital images using parallel computing," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 3, p. 1, 2012.
- [23] A. McAndrew, "An introduction to digital image processing with matlab notes for scm2511 image processing 1 semester 1, 2004," 2004.
- [24] A. Ramezanzad, M. Rezaei, H. Nikmehr, and M. Kalbasi, "Real-time approximate and combined 2d convolvers for fpga-based image processing," *The Journal of Supercomputing*, pp. 1–37, 2023.
- [25] "Reference manual," *Nexys 4 DDR Resource Center*, Sep-2014.