A Mini-Project Report on

FPGA Implementation of Linear Image Filtering

Submitted by

Priyank Srivastava (Roll No. 49)
Shreyash Tiwari (Roll No. 55)
Riddhesh Veling (Roll No. 58)
Aditya Vishwakarma (Roll No. 59)

in partial fulfillment of the requirement for

TE (Semester-VI), Mini Project-2B

in

Department of Electronics & Telecommunication Engineering

under guidance of

Dr. Ravindra Chaudhari

St. Francis Institute of Technology, Mumbai
University of Mumbai
2023-2024

CERTIFICATE

This is to certify that Riddhesh Veling, Shreyash Tiwari, Aditya Vishwakarma, Priyank Srivastava are the bonafide students of St. Francis Institute of Technology, Mumbai. They have successfully carried out the Mini Project-2B titled “FPGA Implementation of Linear Image Filtering” in partial fulfilment of the requirement for TE (Semester-VI), in Electronics and Telecommunication Engineering of Mumbai University during the academic year 2023-2024.

(Internal Examiner)

(External Examiner)

(Dr. Ravindra Chaudhari)
Name of Guide

(Dr. Kevin Noronha)
EXTC HOD

(Dr. Sincy George)
Principal

ACKNOWLEDGEMENT

We are thankful to a number of individuals who have contributed towards our third year project and without their help, it would not have been possible. Firstly, we offer our sincere thanks to our project guide, Dr. Ravindra Chaudhari for his constant and timely help and guidance throughout our preparation.

We are also grateful to the college authorities and the entire faculty especially Mr. Pradeep Ghadi who is our lab assitant for this semester for their support in providing us with the facilities required throughout this semester.

We are also highly grateful to Dr. Kevin Noronha, Head of Department (EXTC), Principal, Dr. Sincy George, and Director Bro. Shantilal Kujur for providing the facilities, a conducive environment and encouragement.

Signatures of all the students in the group

(Riddhesh Veling)

(Aditya Vishwakarma)

(Shreyash Tiwari)

(Priyank Srivastava)

ABSTRACT

The increasing need for real-time image enhancement has driven the exploration of hardware-accelerated solutions, with FPGAs standing out as a promising platform due to their flexibility and computational power. Field-Programmable Gate Arrays (FPGAs) are well-suited for implementing various image enhancement operations due to their parallel processing capabilities, which enable simultaneous processing of multiple pixels, and reconfigurability, allowing for flexible adaptation to different processing tasks. This paper presents a comprehensive study on the implementation of color transformation, brightness adjustment, blurring, edge detection, filter effects, and artistic effects using FPGAs. The implementation is carried out using Verilog HDL within the Xilinx Vivado software suite. Furthermore, the study elucidates the crucial role of COE (coefficient file) generation, which enables the rapid processing of entire pixel values within a single clock pulse, thereby enhancing the speed of operation. Each operation is optimized for FPGA implementation by focusing on reducing latency, which is essential for achieving real-time processing of high-definition images. The study addresses challenges and future directions for improving FPGA-based image processing systems, including algorithm optimization, ultimately advancing their potential for real-time applications across diverse fields.

Contents

List of Figures

List of Tables

List of Abbreviations

FPGA	Field-Programmable Gate Array
VGA	Video Graphics Array
COE	Coefficient File
VHDL	VHSIC Hardware Description Language
HDL	Hardware Description Language
ISE	Integrated Synthesis Environment
I/O	Input/Output
TPU	Tensor Processing Unit
VHSIC	Very High Speed Integrated Circuit
GUI	Graphical User Interface
IDE	Integrated Design Environment
RGB	Red Green Blue color model
CAD	Computer-Aided Deisgn
LUT	Look-Up Tables
FF	Flipflops
BRAM	Block RAM
USB	Universal Serial Bus
GPU	Graphical Processing Unit
CPU	Control Processing Unit

Chapter 1

Introduction

The demand for real-time image processing solutions has escalated, prompting the adoption of Field-Programmable Gate Arrays (FPGAs) as a pivotal platform. This report delves into the benefits of utilizing FPGA technology for real-time image processing, with a specific focus on the Basys3 FPGA framework. Highlighted advantages include high processing speed, parallel processing capabilities, and low latency, which are essential for applications requiring rapid and efficient image enhancement. In various sectors such as medical imaging, surveillance, and remote sensing, image enhancement plays a critical role in improving clarity, contrast, and overall image quality. The Basys3 FPGA framework facilitates a range of image enhancement operations, from convolution-based techniques to color transformation and edge detection, enabling real-time processing and immediate feedback on a VGA display. This introduction sets the stage for exploring FPGA-based real-time image enhancement and its potential impact across diverse industries, from autonomous vehicles to agriculture and augmented/virtual reality experiences.

1.1 Motivation

The adoption of Field-Programmable Gate Arrays (FPGAs) for real-time image processing has gained significant traction due to their ability to deliver high processing speeds, parallel processing capabilities, and low latency. In response to the growing demand for efficient image enhancement solutions across diverse sectors such as medical imaging, surveillance, and remote sensing, this report explores the benefits of FPGA technology, focusing on the Basys3 FPGA framework. By leveraging FPGA-based real-time image processing, this study aims to demonstrate improved system efficiency and functionality. Specifically, we will investigate how FPGA platforms enable rapid and accurate image enhancement operations, showcasing their potential impact on applications like autonomous vehicle navigation, agricultural yield optimization, and immersive augmented/virtual reality experiences. This investigation underscores the importance of FPGA technology in addressing critical challenges and advancing innovation in real-time image processing applications.

1.2 Scope of Project

This project focuses on implementing real-time image processing using FPGA technology, specifically utilizing the Basys3 FPGA framework for image enhancement. It requires a strong understanding of FPGA architecture and programming to leverage their advantages efficiently in real-time tasks. The Basys3 FPGA board will be explored to assess its suitability for executing image processing algorithms effectively.

The project involves selecting and implementing optimized image enhancement algorithms for FPGA, such as convolution-based techniques, color transformation, blurring, sharpening, edge detection, and brightness adjustment. The objective is to utilize FPGA parallel processing capabilities for rapid and efficient enhancement of input images, catering to applications like medical imaging and surveillance.

FPGA hardware will be integrated with software tools like Verilog for configuration and Python for image preprocessing. This integration enables efficient handling and processing of image data, optimizing resource utilization for real-time responsiveness. The goal is to implement these algorithms on Basys3 FPGA, achieving prompt image processing and displaying enhanced results on a VGA display for immediate feedback.

The project also aims to optimize FPGA-based image processing performance by maximizing processing speed, leveraging parallelism, and optimizing memory usage to minimize latency. Through systematic evaluation, the project will assess FPGA-based solutions' efficacy compared to traditional methods, identifying strengths, limitations, and areas for improvement.

By demonstrating FPGA technology's effectiveness in real-time image enhancement and showcasing practical applications across various sectors, this project contributes to advancing FPGA-based solutions in image processing. The insights gained can inform future developments, fostering innovation in real-time FPGA-based image processing applications.

1.3 Organization of Project

- Chapter 2: Summarizing our Literature Survey on this project
- Chapter 3: Detailing Software and Hardware used in this project
- Chapter 4: Detailing the working principle of this project
- Chapter 5: Detailing and Photos of Results acquired.
- Chapter 6: Concluding the project and detailing the future scope.

Chapter 2

Literature Survey

2.1 Literature Review:

The first paper we reviewed was by AlAli et al. titled "Implementing Image Processing Algorithms in FPGA Hardware "[1]. This paper introduces an efficient FPGA-based hardware design for image processing, enhancement, and filtering algorithms. FPGAs are preferred for real-time image processing due to their capability to exploit spatial and temporal parallelism effectively. The proposed approach uses a windowing operator technique to traverse image pixels and apply filters. As image sizes and bit depths increase, software implementations become less practical, requiring real-time hardware systems. The results demonstrate effectiveness with images up to 585x450 pixels, scalable within FPGA memory limits. The design targets the Xilinx Spartan-6 FPGA on a Nexys3 board, offering practical applicability for embedded systems and real-time processing. A brief summary.

- Methodology: This section outlines the hardware implementation, including stepper motors and motor drivers, as well as the electronic components like the Raspberry Pi microcontroller and FPGAs. It also covers the programming aspect, focusing on image processing and algorithm development using Python.
- Result/Conclusion: The experimental results demonstrated the feasibility and effectiveness of FPGA-based hardware design and the paper suggests further research for algorithm enhancement and scalability to larger image sizes.
- Drawbacks/Limitations: The need for improved algorithms for better performance and efficiency. Further research is also needed to scale the current approach for larger image sizes.

The second paper we reviewed was by Said et al. from World Congress on Computer Applications and Information Systems titled "High-level design for image processing on FPGA using Xilinx AccelDSP"[2]. This paper outlines the design and implementation of an image processing application on FPGA using Xilinx AccelDSP to accelerate development from MATLAB high-level descriptions to hardware description language (HDL). The focus is on a FPGA-based architecture for Color Space Conversion, with implementations on Spartan 3A DSP and Virtex 5 devices. Results are presented, discussed, and compared with alternative architectures, highlighting the effectiveness and performance of the proposed FPGA-based approach for image processing tasks. A brief summary:

- Methodology: Involves using MATLAB and Simulink integrated with Xilinx FPGA tools, particularly AccelDSP, to convert MATLAB designs into synthesizable RTL HDL modules. Verification is conducted at each step to ensure accuracy, refining designs from floating-point to fixed-point for hardware implementation.
- Result/Conclusion: The results demonstrate the effectiveness of using Xilinx AccelDSP for rapid prototyping of image processing algorithms on FPGA, contributing to reduced time-to-market and efficient color space conversion from RGB to YCbCr on Spartan 3A DSP and Virtex 5 devices.
- Drawbacks/Limitations: Includes potential complexity in translating MATLAB designs to hardware modules and variability in performance across different FPGA platforms like Spartan 3A DSP and Virtex 5, requiring careful optimization and resource management.

The third paper we reviewed was by Dhanabal R. et al. titled "FPGA Based Image Processing Unit"[3]. This paper explores the application of hardware-based image processing techniques in the medical field, particularly focusing on coin counting using Verilog, a hardware description language. By implementing algorithms for brightness manipulation, operating threshold, and contrast stretching, the study demonstrates an efficient approach compared to conventional edge detection methods for coin counting. The use of Verilog allows for step-wise implementation of these image processing tasks, enhancing system quality and performance. This research highlights the significance of hardware-based solutions in medical multimedia services and underscores the potential benefits of leveraging hardware description languages for image processing applications in healthcare. A brief summary.

- Methodology: Algorithms for brightness manipulation, operating threshold, and contrast stretching were developed. These algorithms were then implemented using Verilog, a hardware description language, to enable hardware-based processing. The Verilog implementation facilitates efficient execution of image processing tasks, enhancing system performance and quality for medical applications.
- Result/Conclusion: This approach simplifies reconfiguration and reconversion of processed images within MATLAB and reduces code complexity compared to traditional edge detection methods, particularly beneficial for circle counting applications.
- Drawbacks/Limitations: Increased hardware complexity due to interfacing requirements with external devices (Bluetooth or serial ports), as well as potential challenges in scalability and adaptability of the FPGA-based implementation to different image processing tasks beyond basic enhancements.

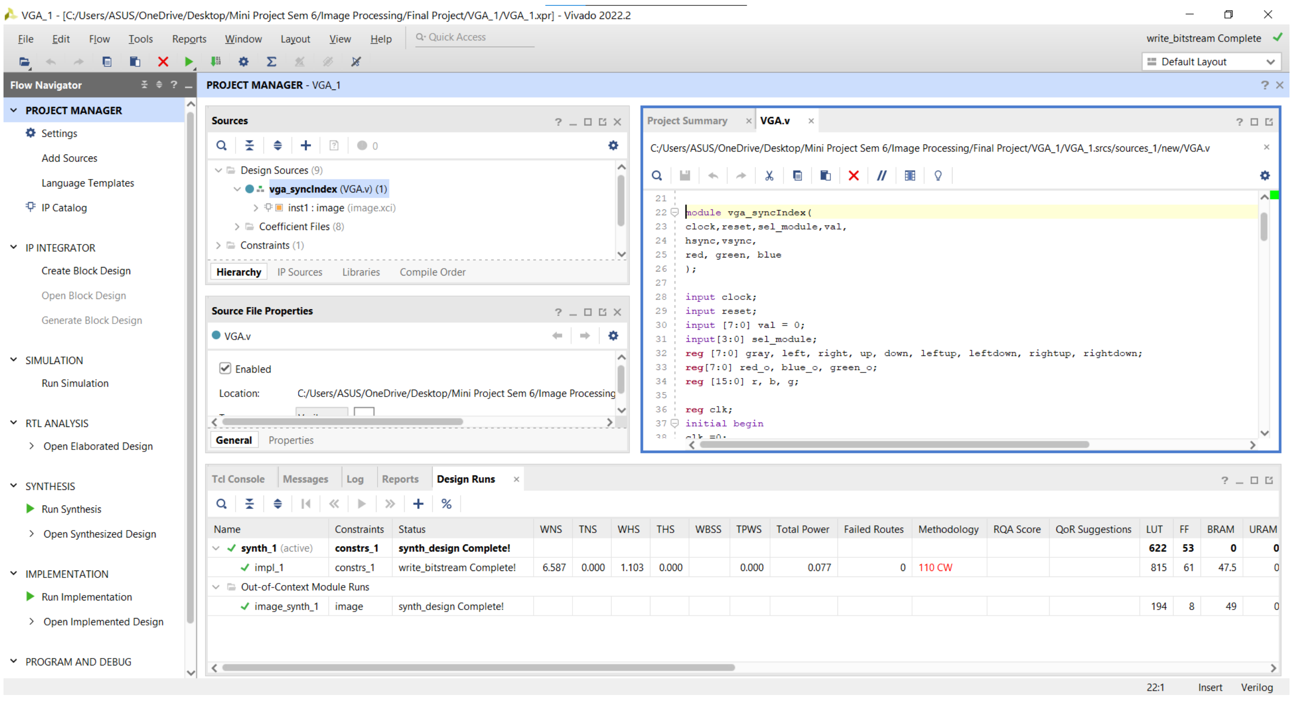
Chapter 3

Software and Hardware Used

3.1 Software

3.1.1 Xilinx Vivado:

Vivado Design Suite[4] is a software suite produced by Xilinx for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis.Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE). We use Xilinx Vivado to plan out our entire project. From writing the verilog code for the project to simulating it using a testbench, synthesizing the project, then we implemented the project on Basys3 Board's Artix-7 FPGA Chip and Finally Generating Bitstream and programming the FPGA using it giving us the hardware result of our project.


VGA_1 - [C:/Users/ASUS/OneDrive/Desktop/Mini Project Sem 6/Image Processing/Final Project/VGA_1/VGA_1.xpr] - Vivado 2022.2

C:/Users/ASUS/OneDrive/Desktop/Mini Project Sem 6/Image Processing/Final Project/VGA_1/VGA_1.srcs/sources_1/new/VGA.v

```verilog

module vga_syncIndex(
clock,reset,sel_module,val,
hsync,vsync,
red, green, blue
);

input clock;
input reset;
input [7:0] val = 0;
input[3:0] sel_module;
reg [7:0] gray, left, right, up, down, leftup, leftdown, rightup, rightdown;
reg[7:0] red_o, blue_o, green_o;
reg [15:0] r, b, g;

reg clk;
initial begin
```

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM
✓ synth_1 (active)	constrs_1	synth_design Complete!												622	53	0	0
✓ impl_1	constrs_1	write_bitstream Complete!	6.587	0.000	1.103	0.000		0.000	0.077	0	110 CW			815	61	47.5	0
Out-of-Context Module Runs																	
✓ image_synth_1	image	synth_design Complete!												194	8	49	0

Figure 3.1: Vivado Workspace UI

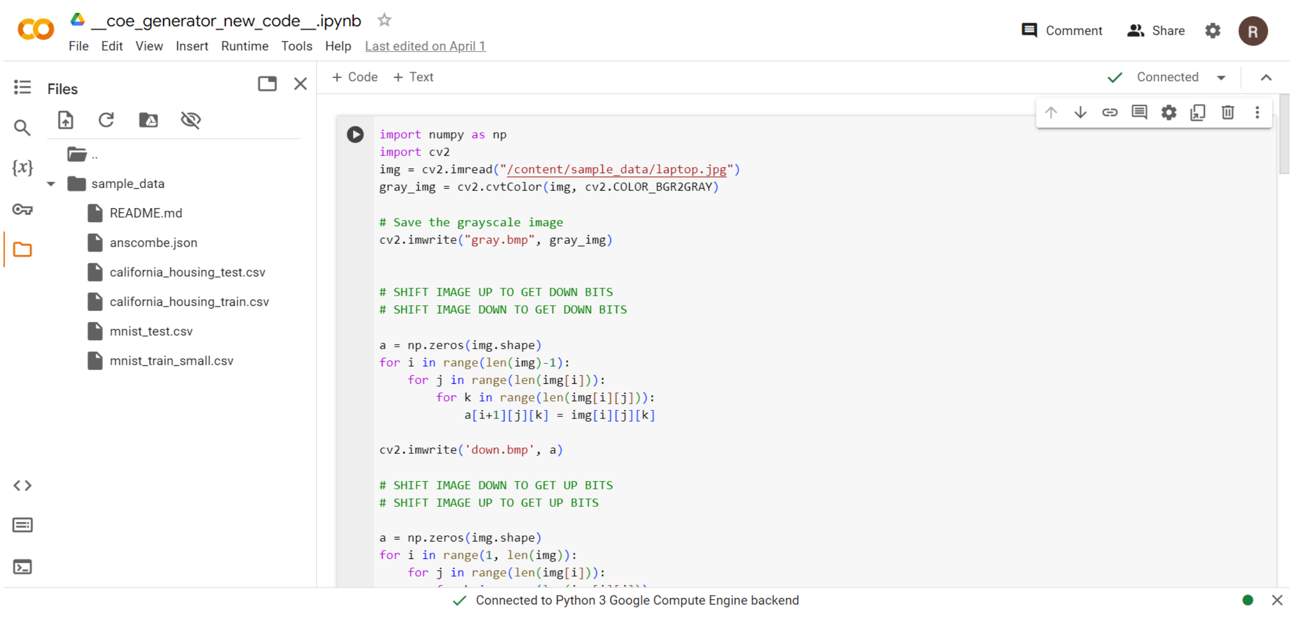
The Vivado simulator is a Hardware Description Language (HDL) simulator that lets you perform behavioral, functional, and timing simulations for VHDL, Verilog, and mixed-language designs. The Vivado simulator environment includes the following key elements:

- xvhdl and xvlog: Parsers for VHDL and Verilog files, respectively, that store the parsed files into an HDL library on disk.
- xelab: HDL elaborator and linker command. For a given top-level unit, xelab loads up all subdesign units, translates the design units into executable code, and links the generated executable code with the simulation kernel to create an executable simulation snapshot.
- xsim: Vivado simulation command that loads a simulation snapshot to effect a batch mode simulation, or a GUI or Tcl-based interactive simulation environment.
- Vivado Integrated Design Environment (IDE): An interactive design-editing environment that provides the simulator user-interface.

In fig.3.1 we can see how a usual workspace on vivado looks like.

3.1.2 Google Colab:

Google Colab[5], short for Google Colaboratory, is a cloud-based platform provided by Google that allows users to write and execute Python code in a browser-based environment. It is designed to facilitate machine learning and data analysis tasks, offering free access to computing resources, including GPU and TPU accelerators, which can significantly speed up computations. Google Colab integrates with Google Drive, enabling users to save and share their Jupyter notebook files directly within Google's ecosystem.



__coe_generator_new_code__.ipynb
File Edit View Insert Runtime Tools Help Last edited on April 1
Comment
Share
R
Files
+ Code
+ Text
Connected
..
sample_data
README.md
anscombe.json
california_housing_test.csv
california_housing_train.csv
mnist_test.csv
mnist_train_small.csv

```
import numpy as np
import cv2
img = cv2.imread("/content/sample_data/laptop.jpg")
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Save the grayscale image
cv2.imwrite("gray.bmp", gray_img)


# SHIFT IMAGE UP TO GET DOWN BITS
# SHIFT IMAGE DOWN TO GET DOWN BITS

a = np.zeros(img.shape)
for i in range(len(img)-1):
    for j in range(len(img[i])):
        for k in range(len(img[i][j])):
            a[i+1][j][k] = img[i][j][k]

cv2.imwrite('down.bmp', a)

# SHIFT IMAGE DOWN TO GET UP BITS
# SHIFT IMAGE UP TO GET UP BITS

a = np.zeros(img.shape)
for i in range(1, len(img)):
    for j in range(len(img[i])):
```

Connected to Python 3 Google Compute Engine backend

Figure 3.2: Google Colab Overview

Google Colab is an open-source project that provides a collaborative environment for coding, similar to Jupyter notebooks, and is especially popular among researchers, students, and data scientists for its ease of use and accessibility. The platform supports libraries like TensorFlow, PyTorch, and scikit-learn, making it suitable for various machine learning applications. Google Colab sessions are temporary but can be saved to Google Drive for later access or sharing. The platform also allows for real-time collaboration, where multiple users can edit and run the same notebook simultaneously, fostering teamwork and knowledge sharing in data analysis and machine learning projects. In fig.3.2 we can see how a usual workspace on Google Colab looks like.

3.2 Hardware

3.2.1 FPGA Board [Basys 3]:

The Basys 3[6] board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. With its high-capacity FPGA (Xilinx part number XC7A35T-1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Basys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs and other I/O devices to allow a large number designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits. The image fig.3.3 shows the top view of the basys board showing all the components of Basys3 board

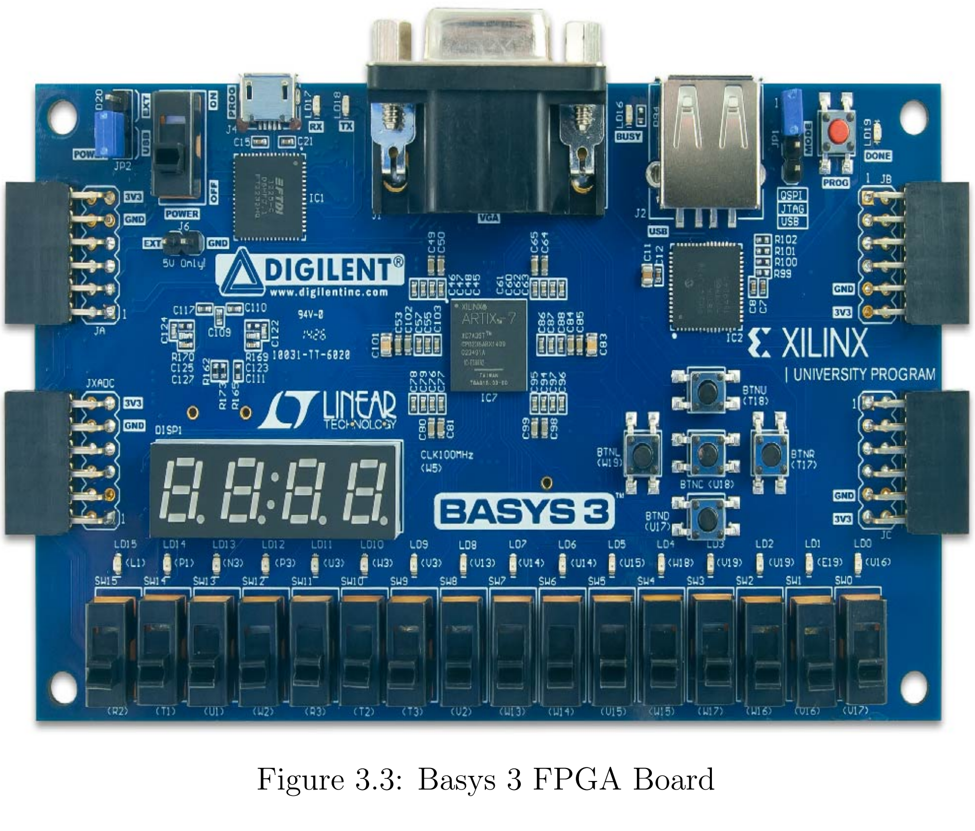


Figure 3.3: Basys 3 FPGA Board

3.2.2 VGA Display:

A VGA (Video Graphics Array)[7] display utilizes multiplexing techniques to control a large number of pixels using fewer pins. Typically organized in rows and columns, each pixel in a VGA display corresponds to an intersection of these rows and columns. Multiplexing reduces the required number of pins, optimizing hardware usage. For instance, an 8x8 display would normally need 65 pins, but with multiplexing, only around 16 pins are needed for standard VGA resolutions like 640x480 with 8-bit color depth. This efficient method supports complex graphics and has contributed to the widespread use of VGA displays in monitors and projectors.

In this project we use the VGA Display along with Basys3 board as shown in fig.3.4

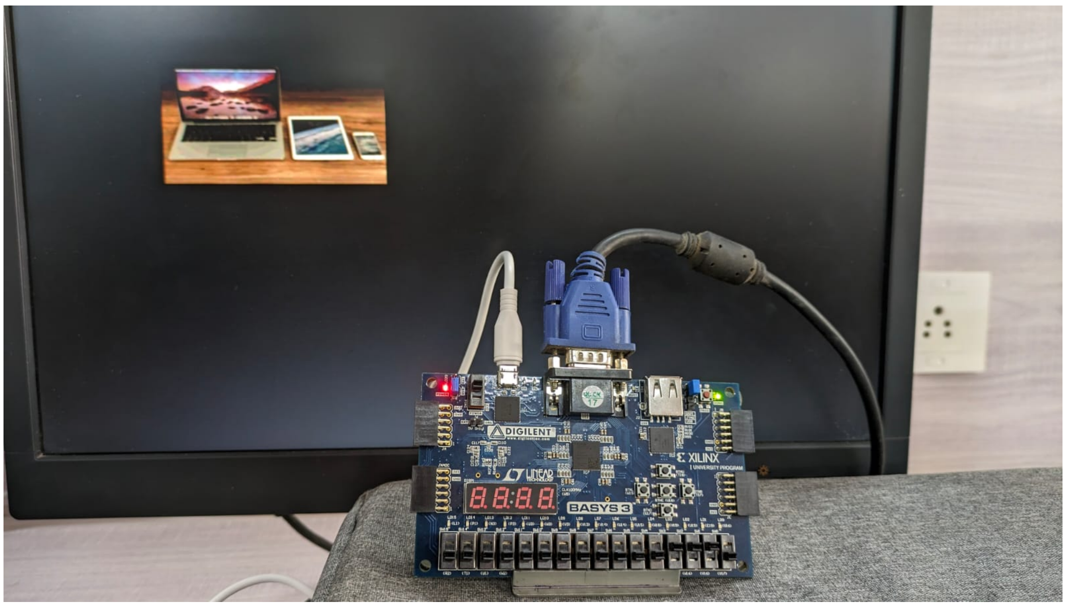


Figure 3.4: VGA Display

Chapter 4

Working Principle

In contrast to conventional methods, the Working Principle employs a novel approach to image input techniques. Rather than following the typical route of converting input images into an intermediate format for processing in a single stage, this innovative method divides the process into two distinct stages. This deliberate segmentation allows for a more nuanced and efficient handling of image data, potentially leading to improved accuracy and performance in subsequent processing tasks. By breaking down the conversion process into two stages, the Working Principle offers a fresh perspective on image processing, emphasizing a more refined and adaptable approach to handling complex visual data

4.1 Initialization

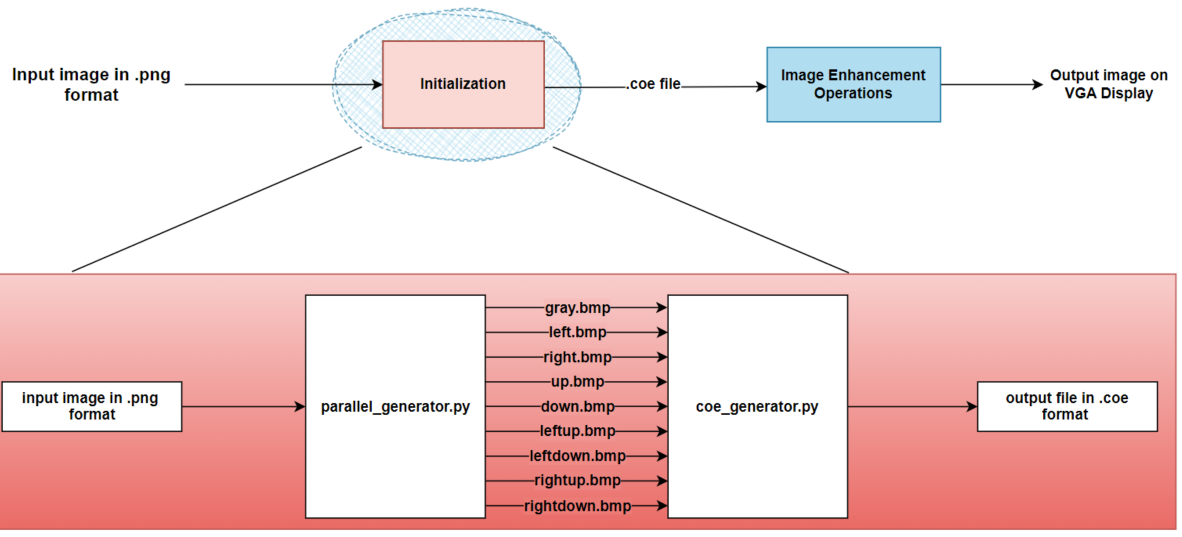


Figure 4.1: Flow Diagram for Initialization stage

4.2 Image Enhancement Operations

The initialization stage consists of 2 python files utilized for converting the input image to a file that can be written to memory for Basys3 FPGA Board. (parallel-image-generator.py) is a python code that splits the given input image to 9 different images.

These images are then used in coe-generator.py python file which creates and initializes the .coe file (coefficient file), setting the necessary preamble to specify the memory initialization radix and indicate the subsequent memory initialization vector. With a list of image names defined for processing, the snippet iterates through each pixel of the grayscale image and its shifted versions, converting their pixel values into binary strings. These binary strings are concatenated and formatted according to the .coe file structure. After processing each row of pixels, the binary data is written to the .coe file, and the string is cleared for the next row. Also, the RGB values are extracted from the original .png image and concatenated at the end of one pixel row. The .coe file is then closed. The .coe file generated in Initialization stage is utilized by the Verilog code. Number of rows in the .coe file are equal to size of image (for example: for an image of size 160x115 we get 18400 rows in .coe file). The .coe file is then loaded into memory. One row of this file is stored into a temporary array at each positive clock edge. Elements of this array are accessed according to the operation selected using switches on Basys3. According to the mode selected, the board executes various image enhancement operations. The resulting processed image is then rendered onto a VGA (Video Graphics Array) display connected to the Basys3, allowing users to visualize the effects of the applied image enhancements in real-time.

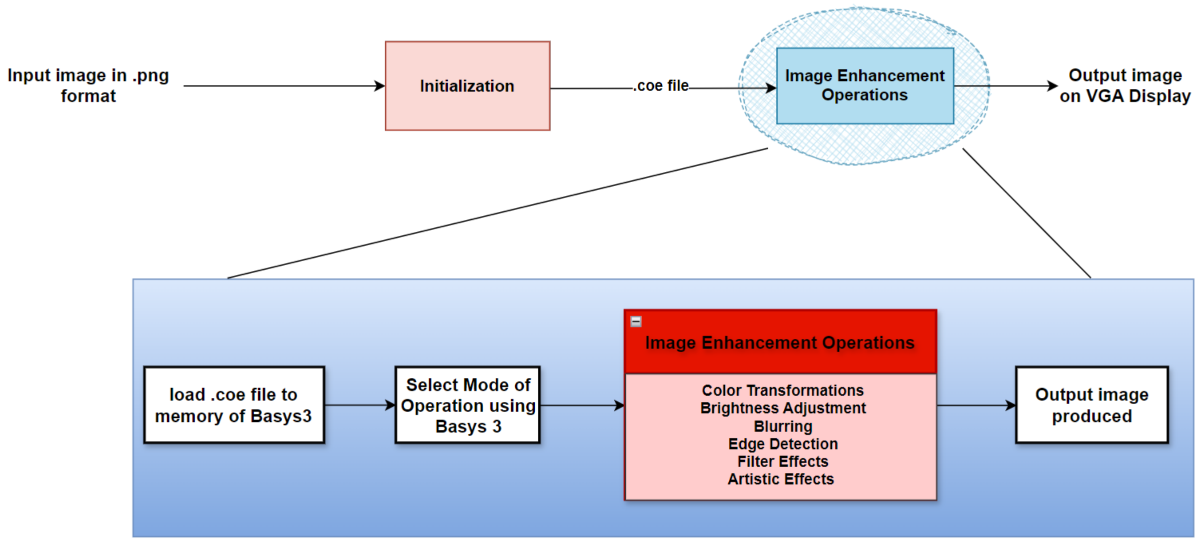

Input image in .png format
Initialization
.coe file
Image Enhancement Operations
Output image on VGA Display
load .coe file to memory of Basys3
Select Mode of Operation using Basys 3
Image Enhancement Operations
Color Transformations
Brightness Adjustment
Blurring
Edge Detection
Filter Effects
Artistic Effects
Output image produced

Figure 4.2: Flow Diagram for Image Enhancement stage

4.3 Implementation

The initialization stage of the Proposed Design Flow involves the utilization of two Python files aimed at facilitating the conversion of input images into a format suitable for writing to memory on the Basys3 FPGA Board. The first file, "parallel-image-generator.py," operates by partitioning the given input image into nine distinct images. These resulting images serve as inputs for the subsequent stage facilitated by the "coe-generator.py" Python script. Within this script, the ".coe" file (coefficient file) is generated and initialized, incorporating essential preamble information to specify the memory initialization radix and indicate the subsequent memory initialization vector. The script iterates through each pixel of the grayscale image and its variously shifted versions which can be seen in fig.4.3.

During this iteration, the pixel values are converted into binary strings. These binary strings are then concatenated and formatted to conform to the structure of the ".coe" file. Following the processing of each row of pixels, the binary data is written to the ".coe" file, and the string is subsequently cleared for the next row. Moreover, during this process, RGB values are extracted from the original ".png" image and appended to the end of each pixel row. Upon completion of this process, the ".coe" file is closed, marking the conclusion of the initialization stage. One line of this .coe file representing a single pixel can be seen in Fig4.4. This meticulous approach ensures the precise conversion and initialization of image data for subsequent FPGA-based processing tasks.

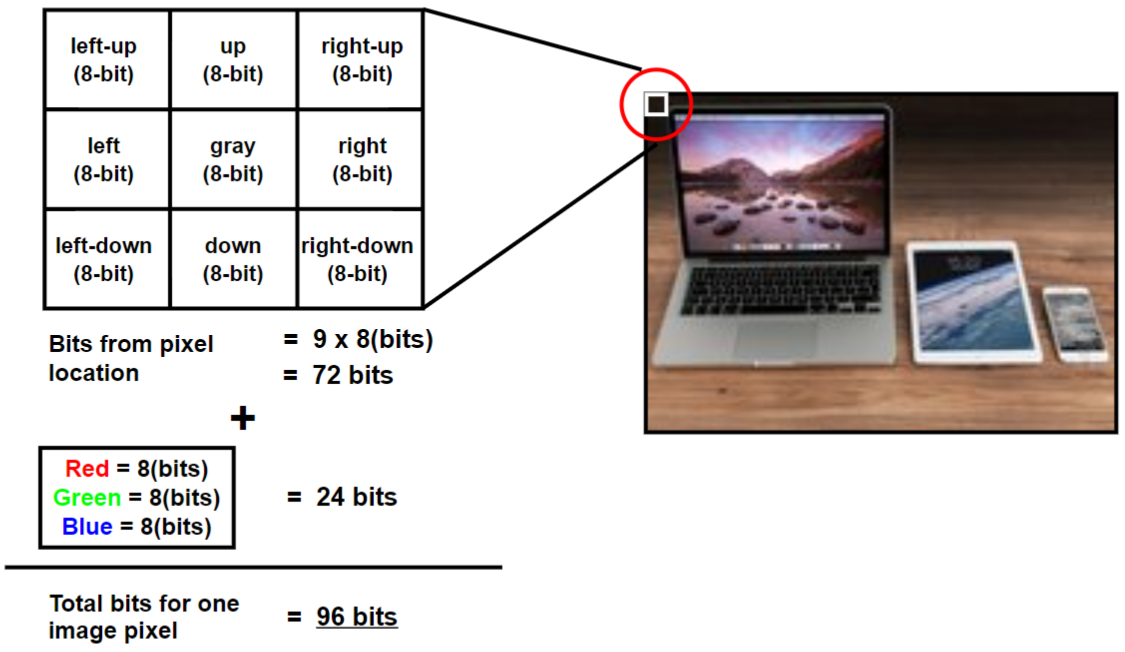


Figure 4.3: Bits formation from one pixel of input image

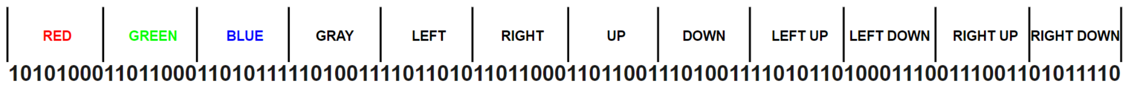


Figure 4.4: One line from the .coe file

4.4 Working of VGA Interface

The VGA (Video Graphics Array) interface facilitates the connection between a computer's graphics card and a display device, typically a monitor. Its functionality hinges on the transmission of analog video signals from the graphics card to the display. These signals encompass various components crucial for generating the visual content, most notably the red (R), green (G), and blue (B) signals, which convey color information for individual pixels. Alongside these RGB signals, VGA incorporates horizontal (HSYNC) and vertical (VSYNC) synchronization signals. HSYNC denotes the end of a line and the start of a new one, while VSYNC indicates the conclusion of a frame and the commencement of the next. Together, these synchronization signals ensure proper alignment and timing of the video data, enabling the display device to accurately render the visual content. By orchestrating the coordinated delivery of RGB signals along with synchronization signals, the VGA interface facilitates the seamless transmission and display of images on compatible monitors or screens.

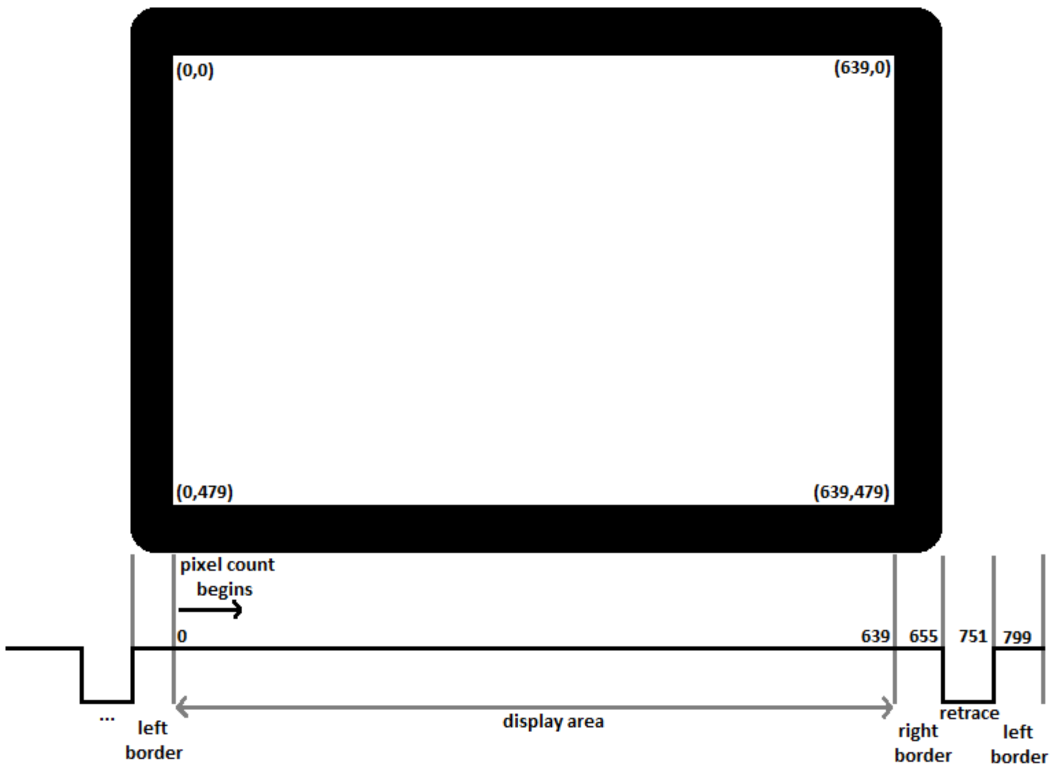


Figure 4.5: VGA Interface

4.5 Image Enhancement Operations

4.5.1 Color Transformation

RGB to Gray

Converting a color image to grayscale involves splitting each 24-bit pixel into red, green, and blue components. By adjusting these channels using right shift operations to represent 28.1%, 56.2%, and 9.3% of their respective colors [10], the color intensities are redistributed to create a balanced grayscale representation. The adjusted values are then combined to create an 8-bit grayscale representation for each pixel, allowing the entire image to be converted to grayscale.

$$Gray_{op} = (R \gg 2) + (R \gg 5) + (G \gg 1) + (G \gg 4) + (B \gg 4) + (B \gg 5)$$

Color Inversion

Color inversion in image processing involves transforming the RGB components of each pixel in a color image to their respective inverted values. This is achieved by subtracting the original intensity values of the red, green, and blue channels from 255. This operation reverses the color scheme of the image, making bright areas dark and vice versa. Applying this technique to all pixels ensures a uniform and complete inversion of colors, enhancing the overall image. In this context, R, G, and B represent the input RGB components, and R_{op}, G_{op}, and B_{op} represent the output components.

$$R_{op} = 255 - R$$
$$G_{op} = 255 - G$$
$$B_{op} = 255 - B$$

RGB Filters

The Red, Green, and Blue image filtering techniques use RGB values to highlight specific primary colors in an image. The Red filter preserves red intensity values while setting green and blue values to zero, emphasizing areas with a lot of red. The Green filter keeps green intensity values and removes red and blue, highlighting green-dominated regions. The Blue filter maintains blue intensity values and removes red and green, drawing attention to areas with blue hues. This method reduces pixels not matching the selected color, focusing on primary RGB colors throughout the image.

4.5.2 Brightness Adjustment

Increase Brightness

The Increase Brightness technique employs a simple yet effective method to enhance the luminosity of an image uniformly. By adding a predetermined adjustment factor $'g'$ to the intensity of each pixel, the technique effectively brightens the entire image. The formula ensures that intensity values remain within the valid range of 0 to 255. Here, $I_k(r, c)$ and $J_k(r, c)$ are input and output pixel values respectively.

$$J_k(r,c) = \begin{cases} I_k(r,c) + g & \text{,if } I_k(r,c) + g \leq 255 \\ 255 & \text{,if } I_k(r,c) + g > 255 \end{cases}$$

Decrease Brightness

The Decrease Brightness technique operates by uniformly reducing the luminance of an image. his method diminishes the intensity of each pixel by a predetermined adjustment factor $'g'$. By systematically lowering the brightness across the image, the technique effectively attenuates the luminosity of brighter regions while preserving the visual details and contrast within the image.

$$J_k(r,c) = \begin{cases} I_k(r,c) - g & \text{,if } I_k(r,c) - g \geq 0 \\ 0 & \text{,if } I_k(r,c) - g < 0 \end{cases}$$

4.5.3 Edge Detection Filters

Laplacian Filter

Edge detection utilizing the Laplacian Filter is pivotal in image processing, as it allows for the precise localization of edges and contours. By amplifying the intensity of the central pixel and subtracting the weighted sum of neighboring pixel intensities, this method effectively highlights areas of rapid intensity changes. The Laplacian Filter is particularly adept at detecting edges regardless of their orientation or thickness, making it a versatile tool for edge detection tasks in various image processing applications. However, it has certain drawbacks, including sensitivity to noise, tendency to produce thick edges, and susceptibility to gradient reversal issues. The Laplacian Filter kernel can be seen in Fig 4.6 (a) and (b).

Sobel Edge Detection

The Sobel Edge Detection algorithm is a fundamental technique in image processing aimed at highlighting edges within an image. This method involves the application of the Sobel operator in both the horizontal (x) (shown Fig 4.6 (c)) and vertical (y) (shown Fig 4.6 (d)) directions to detect changes in intensity indicative of edges . By convolving the image with separate Sobel kernels for x and y directions, the algorithm computes the gradient magnitude of each pixel, representing the rate of change of intensity. Subsequently, the root mean square (RMS) value of the gradient magnitudes in both directions is calculated to create the final edge-emphasized image. This approach effectively enhances edges while suppressing noise, enabling precise edge detection and boundary delineation in various image processing applications.

0	-1	0
-1	4	-1
0	-1	0

(a)

-1	-1	-1
-1	8	-1
-1	-1	-1

(b)

-1	-2	-1
0	0	0
1	2	1

(c)

-1	0	1
-2	0	2
-1	0	1

(d)

Figure 4.6: Edge Detection kernels (a) Laplacian kernel (used for horizontal and vertical edge differences), (b) Laplacian kernel , (c) Gx horizontal component, (d) Gy vertical component

4.5.4 Blurring Filters

Average Blurring

Average blurring is a widely used image filtering technique employed to reduce noise and smooth out images. This method involves convolving the image with a kernel where each element represents a weighted average of its neighboring pixels. Typically, the kernel, as shown in Fig 4.7 (a), is a square matrix with equal weights assigned to each element, resulting in a uniform blur effect across the image. The size of the kernel determines the extent of blurring, with larger kernels producing stronger smoothing effects but potentially sacrificing image sharpness. Average blurring is effective in reducing high-frequency noise while preserving overall image structure, making it a valuable tool in various image processing applications such as denoising and preprocessing for further analysis.

Motion Blurring (xy)

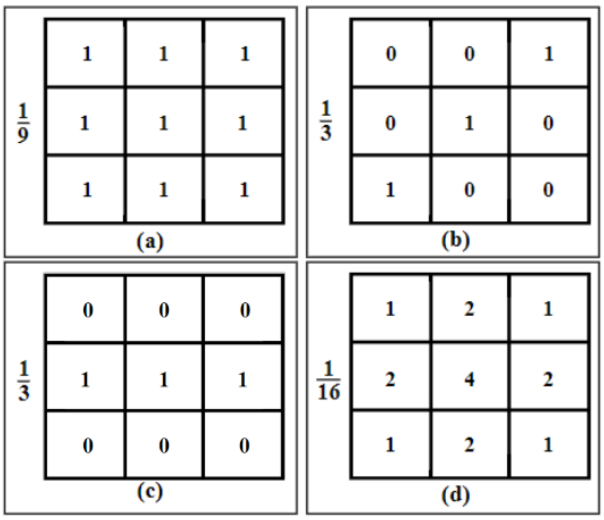
Motion blurring, often employed in image processing and computer graphics, simulates the effect of an object in motion by averaging the pixel values along a specific direction. In the case of motion blurring in both the x and y directions (xy), the kernel used is typically a square matrix with non-zero elements arranged diagonally. Each element of the kernel represents the weight assigned to the corresponding pixel in the image, with higher weights typically assigned to pixels closer to the direction of motion. As a result, motion blurring in the xy direction creates streak-like artifacts in the image, mimicking the appearance of objects moving horizontally and vertically. Fig 4.7 (b) kernel is used for Motion Blurring in XY direction.

Motion Blurring (x)

Motion blurring in the x-direction, a prevalent technique in image processing, mimics the effect of objects moving horizontally. It involves convolving the image with a kernel that averages pixel values along the x-axis, thereby generating streak-like artifacts in the direction of motion. The kernel typically consists of a row with non-zero elements representing the weights assigned to neighboring pixels. Pixels closer to the motion direction are assigned higher weights, intensifying the blurring effect.

Gaussian Blurring

Gaussian blurring is a widely utilized image processing technique designed to reduce noise and smooth out images while preserving important features. This method involves convolving the image with a Gaussian kernel, which is characterized by a bell-shaped curve [12] representing the distribution of pixel weights. Fig 4.7 (d) kernel's central pixel carries the highest weight, gradually decreasing as it moves away from the center. This results in a gentle and isotropic blurring effect across the image, effectively reducing high-frequency noise.



$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

(a)

$\frac{1}{3}$

0	0	1
0	1	0
1	0	0

(b)

$\frac{1}{3}$

0	0	0
1	1	1
0	0	0

(c)

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

(d)

Figure 4.7: Blurring Kernels (a) Average Blurring kernel, (b) Motion Blurring xy kernel, (c) Motion Blurring x kernel, (d) Gaussian Blurring kernel

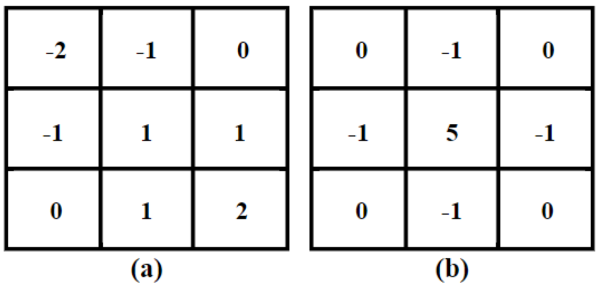
4.5.5 Artistic Effects Filters

Embossing

Embossing is a popular method for image enhancement that gives the impression of three dimensions by highlighting edges in it. This technique entails convolving the picture using an emboss kernel, which is usually a little square matrix. The kernel's symmetrically organized positive and zero-valued parts surround a negative-valued core. This kernel emphasizes edges by contrasting the pixel brightness of adjacent regions when applied to the picture.

Sharpening

Sharpening is a vital image enhancement technique aimed at improving image clarity and detail by enhancing edge contrast. This process involves convolving the image with a sharpening kernel, which accentuates differences in pixel intensities between neighboring regions. Typically, the sharpening kernel consists of a central element with a positive value, surrounded by negative and zero-valued elements arranged symmetrically. As a result, when applied to the image, the sharpening kernel intensifies the edges, making them appear more defined and prominent. However, this process can also amplify noise and artifacts present in the image.



-2	-1	0
-1	1	1
0	1	2

(a)

0	-1	0
-1	5	-1
0	-1	0

(b)

Figure 4.8: (a) Embossing, (b) Sharpening

Chapter 5

Results and Analysis

5.1 Experimental Results

We have used Artix-7 Basys3 FPGA Board for synthesis purpose. Within the Basys3 FPGA framework, users can choose from various image enhancement operations using physical switches/buttons, including filters, color transformations, and blurring. The 4 switches represent 4 bits for each image enhancement mode. Toggling switches/buttons enables smooth transitions between modes, facilitating customization of image processing workflows, as demonstrated in the presented outcomes.

Table 5.1: Bit Selection for Operations

Fig No.	**Enhancement Operation**	**Bits (On Basys3)**
Fig 5.1 (a)	RGB to Gray	0000
Fig 5.1 (b)	Increase Brightness	0001
Fig 5.1 (c)	Decrease Brightness	0010
Fig 5.1 (d)	Color Inversion	0011
Fig 5.2 (a)	Red Filter	0100
Fig 5.2 (b)	Green Filter	0101
Fig 5.2 (c)	Blue Filter	0110
Fig 5.2 (d)	Original Image	0111
Fig 5.3 (a)	Average Blurring	1000
Fig 5.3 (b)	Sobel Edge Detection	1001
Fig 5.3 (c)	Outline (Laplacian)	1010
Fig 5.3 (d)	Motion Blurring (xy)	1011
Fig 5.4 (a)	Emboss	1100
Fig 5.4 (b)	Sharpen	1101
Fig 5.4 (c)	Motion Blurring (x)	1110
Fig 5.4 (d)	Gaussian Blurring	1111

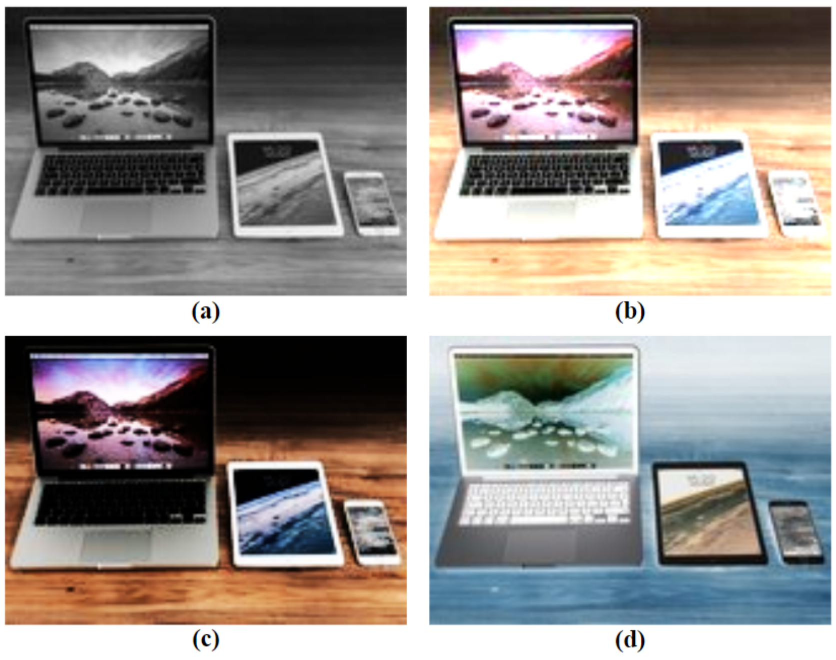


Figure 5.1: (a) RGB to Gray, (b) Increase Brightness, (c) Decrease Brightness, (d) Color Inversion

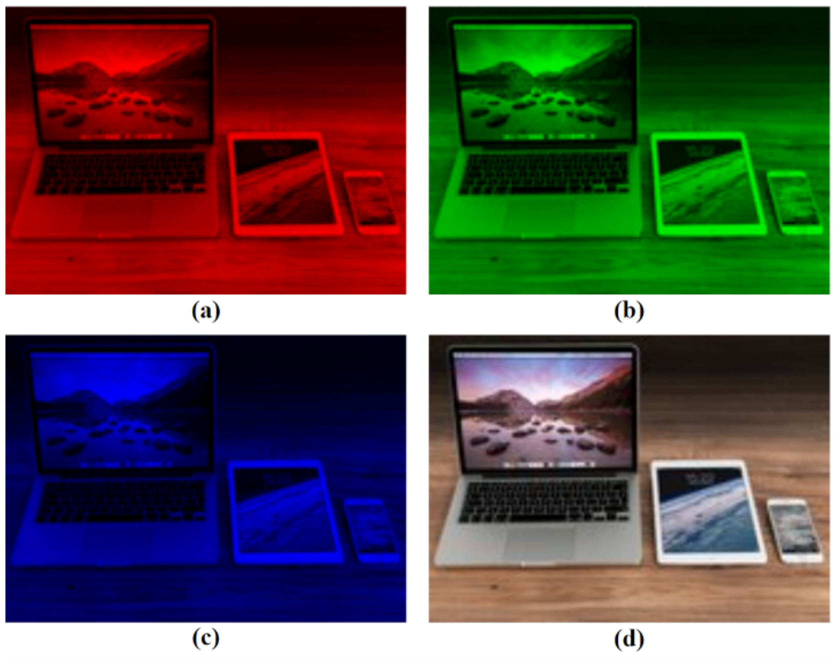


Figure 5.2: (a) Red Filter, (b) Green Filter, (c) Blue Filter, (d) Original image

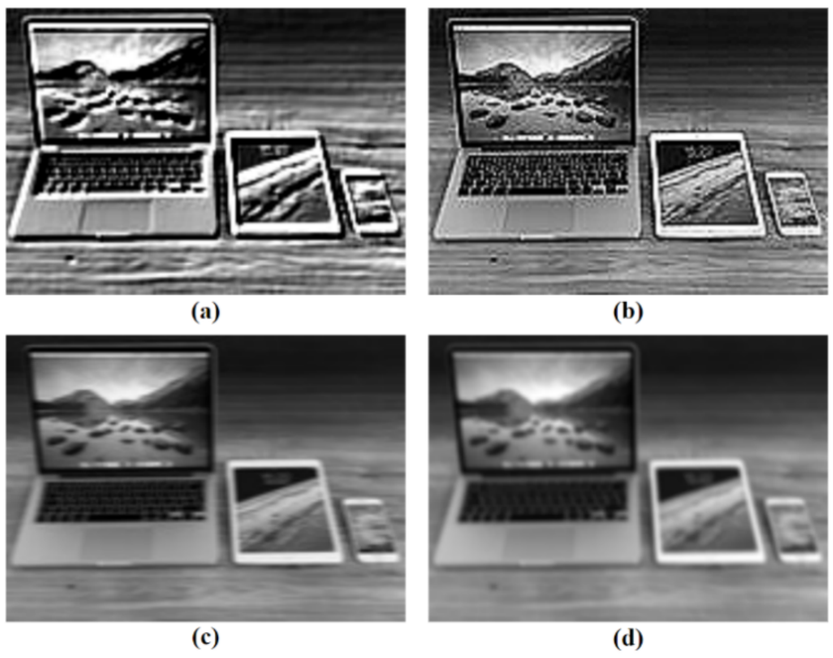


Figure 5.3: (a) Average Blurring, (b) Sobel Edge Detection, (c) Outline(Laplacian Edge Detection), (d) Motion Blurring (xy)

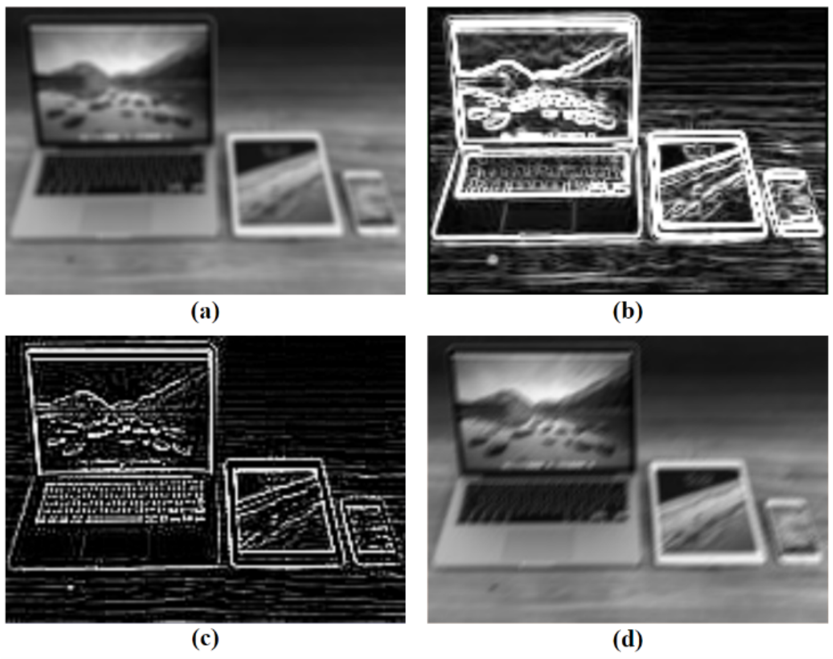


Figure 5.4: (a) Embossing, (b) Sharpening, (c) Motion Blurring (x), (d) Gaussian Blurring

Table 5.2: Hardware Resource Utilization Summary

Resource	**Utilization**	**Available**	**Utilization %**
LUT	838	20800	4.03
FF	61	41600	0.15
BRAM	49	50	98.00
IO	20	106	18.87

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Implementation of FPGA for image enhancement is a Verilog-based project designed for the Basys3 FPGA, aimed at facilitating image enhancing operations such as convolution on input images. It leverages Block RAM to store images and displays them through a VGA interface. Development is carried out using Verilog and Python, with the Vivado software suite. The project offers two primary implementations: one for the transfer of images between a PC and the FPGA, and another for the display of processed images on a monitor. The VGA interface is specifically configured for a 480p display with a 60Hz refresh rate. To initialize the block RAM in Xilinx FPGA designs, a .coe file is provided. The project's development structure supports basic image enhancement and convolution-based operations, each requiring specific implementation approaches. Python scripts are utilized to generate .coe files tailored to these operations. Additionally, incorporating hardware accelerators or co-processors could further enhance the system's performance and enable it to tackle more computationally intensive tasks.

6.2 Future Scope

The future scope of FPGA implementation of linear image filtering lies in the development of more efficient and specialized algorithms for specific tasks, as well as the optimization of existing methods for FPGA implementation. This includes the exploration of novel filtering techniques, such as guided filters, which can provide improved results in noise reduction and detail smoothing while maintaining edge preservation. Furthermore, the integration of FPGA-based image processing systems with other hardware and software platforms, such as USB-connected FPGA boards and Matlab GUIs, can expand their applicability and ease of use in various fields, including computer vision, computer graphics, and medical imaging.

We also can implement non-linear filters. Nonlinear filters are widely used in image processing applications to remove impulse noise and preserve edges. They are particularly useful when linear filters, such as Wiener filters, tend to blur the edges, do not remove impulsive noise effectively, and do not perform well in the presence of non-Gaussian noise. Nonlinear filters, such as median filters, are known to remove impulse noise and preserve edges, making them a popular choice for image restoration techniques.

It also presents a parallel FPGA implementation of image convolution, which is a common algorithm in graphics editors. The thesis investigates and presents a way to implement the algorithm with 16 parallel convolutions, achieving better performance than multi-threaded implementations on both GPU and CPU.

Bibliography

[1] Mohammad I. AlAli, Khaldoon M. Mhaidat, and Inad A. Aljarrah. Implementing image processing algorithms in fpga hardware. *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, Dec 2013.

[2] Yahia Said, Taoufik Saidani, and Mohamed Atri. High-level design for image processing on fpga using xilinx acceldsp. *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, Jan 2014.

[3] Dhanabal R, Sarat Kumar Sahoo, Bharathi V, Bh.S.R. Phanindra Varma, D. Kalyan, and V. Divya. Fpga based image processing unit. *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, Jan 2015.

[4] Vivado Overview. `https://www.xilinx.com/products/design-tools/vivado.html`, April 2024. [Online; accessed 18. Feb. 2024].

[5] Google Colaboratory. `https://colab.research.google.com/?hl=en-GB`, April 2024. [Online; accessed 18. Feb. 2024].

[6] Digilent. Basys 3. `https://digilent.com/reference/programmable-logic/basys-3/start`, 2023. [Online; accessed 18. Feb. 2024].

[7] Contributors to Wikimedia projects. Video Graphics Array - Wikipedia. `https://en.wikipedia.org/w/index.phptitle=Video_Graphics_Array&oldid=1213264842`, March 2024. [Online; accessed 18. Feb. 2024].