

DE-41 (EE)

AYESHA,

MUHAMMAD HASSAAN,

ALISHBA

**CO-PROCESSOR DESIGN FOR IMAGE
ENHANCEMENT USING FPGA**



**COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND
TECHNOLOGY RAWALPINDI**

2023

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING



PROJECT REPORT

DE-41 EE

**CO-PROCESSOR DESIGN FOR IMAGE ENHANCEMENT
USING FPGA**

Submitted to the Department of Electrical Engineering in partial fulfillment of the
requirements for the degree of

Bachelor of Engineering

in

Electrical

2023

Sponsoring DS:

Submitted By:



CERTIFICATE OF APPROVAL

It is to certify that the project “**CO-PROCESSOR DESIGN FOR IMAGE ENHANCEMENT USING FPGA**” was done by **NC Ayesha Javaid**, **NC Muhammad Hassaan Aftab**, and **NC Alishba Zulfiqar** under supervision of **Dr. Usman Ali**.

This project is submitted to **Department of Electrical Engineering**, College of Electrical and Mechanical Engineering (Peshawar Road Rawalpindi), National University of Sciences and Technology, Pakistan in partial fulfilment of requirements for the degree of Bachelor of Electrical Engineering.

Students:

1. Ayesha Javaid

NUST ID: _____ Signature: _____

2. Muhammad Hassaan Aftab

NUST ID: _____ Signature: _____

3. Alishba Zulfiqar

NUST ID: _____ Signature: _____

APPROVED BY:

Project Supervisor: _____ Date: _____

Dr. Usman Ali

DECLARATION

We affirm that the content presented in this Project Thesis is original and has not been submitted in support of any other degree or qualification at this or any other educational institution. We acknowledge that any act of plagiarism will result in full responsibility and may lead to disciplinary action, including the potential cancellation of our degree, based on the severity of the offense.

1. Ayesha Javaid _____

2. Muhammad Hassaan Aftab _____

3. Alishba Zulfiqar _____

COPYRIGHT STATEMENT

The student author's intellectual property and copyright cover the text of this thesis. Any copies or excerpts from this thesis should adhere to the author's guidelines precisely and be stored in the NUST College of E&ME Library. Additional copies of such copies may only be made with the author's written consent.

Except when otherwise noted, all intellectual property rights in connection with the content presented in this thesis are owned by NUST College of E&ME and may not be used by third parties without the College's prior written consent. The College of E&ME will set the terms and circumstances of such agreements. Please contact the library of the NUST College of E&ME in Rawalpindi for more details on disclosure and exploitation conditions.

ACKNOWLEDGEMENTS

First and foremost, we want to thank Allah Ta'ala for giving us the intelligence and bravery to understand and overcome the difficulties encountered during this difficult attempt. The direction and assistance of many people are crucial to any project's success, in addition to our combined efforts. We would like to express our profound gratitude to our supervisor, Dr. Usman Ali, whose outstanding help, encouragement, and direction were crucial to our success.

Additionally, we owe a debt of gratitude to our cherished parents, whose unfailing support and tolerance were essential to our path. They have continuously supported us when we have needed them.

Finally, we would like to thank the Electrical Engineering Department for supporting us throughout our academic careers and giving us the opportunity to succeed in our area. We would like to express our gratitude to the instructors and support personnel who have worked tirelessly to provide us with first-rate resources and direction.

ABSTRACT

The design and creation of a co-processor for image enhancement is demonstrated in this project. The procedure entails creating algorithms in MATLAB, designing hardware in Verilog, creating software in C, and integrating the system in Vivado. The implementation of the histogram equalization and fixed-point conversion algorithms is part of the MATLAB phase. The development of the BRAM module and histogram computation for the programmable logic (PL) component are the main goals of the hardware design phase. On the other hand, the C programming phase of software design concentrates on fixed point conversion and histogram equalization for the processing system (PS). Vivado is used to integrate the IP cores into the system, allowing for seamless communication between the hardware and software elements.

SUSTAINABLE DEVELOPMENT GOALS

The initiative of developing a co-processor for image enhancement is in line with SDG 9, which focuses on resilient infrastructure, sustainable industrialization, and innovation.

The project's goal is to increase the efficiency and performance of image processing activities by establishing a specialized co-processor for image enhancement. By offering specific hardware capabilities that may expedite image enhancing algorithms, this helps to the creation of sophisticated infrastructure. Furthermore, the project encourages creativity by investigating co-design approaches that make use of both hardware and software components. It provides quicker and more accurate image enhancement by improving the co-processor architecture, opening up new possibilities in domains like digital photography, computer vision, and multimedia applications.

Furthermore, the project promotes sustainable industrialization by concentrating on the development of an energy-efficient co-processor that increases computing efficiency while using the least amount of resources. This is consistent with the objective of developing sustainable technologies that decrease environmental impact and encourage resource stewardship.

TABLE OF CONTENTS:

Contents

CERTIFICATE OF APPROVAL	i
COPYRIGHT STATEMENT:	iii
ACKNOWLEDGMENTS	iv
ABSTRACT	v
SUSTAINABLE DEVELOPMENT GOALS	vi
TABLE OF CONTENTS:	vii
Table of Figures:	x
LIST OF ABBREVIATIONS	xii
1. INTRODUCTION	1
1.1. Project Overview:	1
1.2. Problem Statement:	1
1.3. Approach	2
1.3.1. Algorithm Design:	2
1.3.2. Fixed-Point Conversion:	2
1.3.3. Verilog and C Integration:	3
1.3.4. FPGA Board Selection:	3
1.3.5. Integration:	3
1.3.6. Testing:	3
1.4. Objectives:	3
1.5. Specifications:	3
1.6. Deliverables:	4
1.7. Organization of Thesis:	4
2. Literature Review	7
2.1. Background	7
2.2. Introduction to FPGA	7
2.2.1. Arty Z7-10:	8
2.3. FPGA Design and Programming	9
2.4. Image Processing	10
2.4.1. Image Enhancement:	10
2.5. Histogram Equalization	13
2.5.1. Methodology	15

2.5.2.	Mathematical concept	16
2.5.3.	Impact of Histogram Equalization on Final Output:	16
2.5.4.	Pseudo code implementation:.....	18
2.6.	Existing Literature Related to the topic:	18
2.7.	Problem Formulation of the Topic:	19
3.	Chapter 3: Design and Development	21
3.1.	Design Flow:	21
3.2.	Hardware and Software part:	24
3.3.	Software part:	24
3.3.1.	Designing algorithm.....	25
3.3.2.	Hex File Generation using MATLAB:.....	26
3.4.	Hardware designing using HDL.....	27
3.4.1.	BRAM module	27
3.4.2.	Histogram Module	27
3.5.	Software part using C:.....	30
3.6.	Software/Hardware integration:	31
3.7.	Hardware Interfacing	32
3.8.	Hardware Requirement:	32
3.9.	Software Requirement:.....	32
4.	Result Analysis	35
4.1.	Algorithm Design on MATLAB	35
4.1.1.	Hex file generation:.....	36
4.2.	Hardware Result Analysis: Verilog Implementation	38
4.2.1.	Block RAM module	38
4.2.2.	Histogram Module	42
4.3.	Software result Analysis: C Implementation	44
4.3.1.	Histogram Implementation.....	44
4.3.2.	Histogram Equalization.....	46
4.3.3.	Mean Square Error	49
4.4.	IP core Integration.....	49
4.4.1.	Bitstream generation	51
4.5.	Hardware Interfacing	51
5.	Chapter 5: Future Work	54
5.1.	Real-Time Image Enhancement:.....	54
5.2.	Optimization for Parallel Processing:	54

5.3. Integration with Machine Learning Techniques:	54
5.4. Advanced Image Enhancement Algorithms:.....	54
5.5. Energy Efficiency and Power Optimization:	55
5.6. System Integration and Deployment:.....	55
6. Chapter 6: Conclusion.....	57
6.1. Overview	57
6.2. Objective Achieved/ Achievements	57
6.3. Contributions.....	57
6.4. Limitations	58
6.5. Applications	58
6.5.1. Medical Imaging:	58
6.5.2 Computer Vision:.....	58
6.5.3. Remote Sensing:.....	58
6.5.4. Video Processing:.....	58
6.5.5. Embedded Systems:	59
REFERENCES.....	61

Table of Figures:

Figure 1 : Arty Z7	7
Figure 2: ARM core Processor.....	8
Figure 3 : Original Image.....	11
Figure 4 : Enhanced Image	11
Figure 5 : Point Processing	12
Figure 6 : Neighborhood Processing.....	12
Figure 7 : Global Processing	13
Figure 8 : Original Image.....	14
Figure 9 : Histogram Image	14
Figure 10 : MATLAB Flow Chart	16
Figure 11 : Original Image Histogram	17
Figure 12 : Equalized Image Histogram	17
Figure 13 : Pseudo Code	18
Figure 14 : Project Block Diagram	24
Figure 15 : MATLAB Algorithm.....	26
Figure 16: BRAM	27
Figure 17 : Histogram Block Diagram.....	30
Figure 18 : Histogram Display.....	35
Figure 19 : Histogram Equalization using Built-in function.....	36
Figure 20 : Command Window.....	37
Figure 21: Hex File	38
Figure 22 : B-RAM simulation	39
Figure 23 : Console Window	40
Figure 24 : RTL diagram of BRAM	41
Figure 25 : Histogram Module Simulation	42
Figure 26 : RTL diagram	43
Figure 27 : Simulation Console Result	43
Figure 28 : Histogram generated by MATLAB.....	45
Figure 29 : Histogram generated by C in vivado	46
Figure 30 : MATLAB output.....	47
Figure 31 : VIVADO output	47

Figure 32 : Synthesis Report	48
Figure 33 : Simulation Console Window	48
Figure 34 : IP core Integration	50
Figure 35 : Address Editor	51

LIST OF ABBREVIATIONS

FPGA	Field Programmable Gate Array
PL	Programmable Logic
PS	Processing System
ASIC	Application Specific Integrated Circuits
CLB	Configurable Logic Block
SoC	System on Chip
APSoC	All Programmable System on Chip
ARM	Advanced RISC Machine
USB	Universal Serial Bus
HDMI	High-Definition Multimedia Interface
I/O	Input/Output
HDL	Hardware Description Language
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
RTL	Register Transfer Level
CLAHE	Contrast Limited Adaptive Histogram Equalization
AHE	Adaptive Histogram Equalization
CDF	Cumulative Distribution Function
USB-UART	Universal Serial Bus - universal asynchronous receiver / transmitter
HDMI	High-Definition Multimedia Interface
IP	Internet Protocol
PS	Processing System

RGB	Red Green Blue
MSE	Mean Square Error
ASCII	American Standard Code for Information Interchange
PL	Programmable Logic
HDL	High-Definition Language
BRAM	Block RAM
I/O	Input/Output
FSM	Finite State Machine
RAM	Random Access Memory.
MUX	Multiplexer

CHAPTER # 01

INTRODUCTION

- 1.1. PROJECT OVERVIEW
- 1.2. PROBLEM STATEMENT
- 1.3. APPROACH
 - 1.3.1. ALGORITHM DESIGN
 - 1.3.2. FIXED POINT CONVERSION
 - 1.3.3. VERILOG AND C INTEGRATION
 - 1.3.4. FPGA BOARD SELECTION
 - 1.3.5. INTEGRATION
 - 1.3.6. TESTING
- 1.4. OBJECTIVES
- 1.5. SPECIFICATIONS
- 1.6. DELIVERABLES
- 1.7. ORGANIZATION OF THESIS

INTRODUCTION

This introduction provides a brief description of the research, starting with a critical examination of the limits of existing image enhancing approaches. It then specifies the project's goals, limitations, and requirements, as well as the deliverables that are intended to be generated. Finally, the chapter finishes with a thesis organizational framework.

1.1. Project Overview:

This research focuses on developing an image improvement coprocessor utilizing FPGA technology, with a particular emphasis on histogram equalization. For the hardware and software components, Verilog and C were utilized, respectively, while Vivado was used for integration.

The image's histogram was created in Verilog, and histogram equalization was done in C. Before comparing the results from the two platforms, the fixed-point conversion (Q16.16) [1] was conducted in both MATLAB and Vivado.

During the testing phase, the results were compared to confirm the coprocessor's accuracy and efficacy. The comparison of MATLAB and Vivado results revealed the accuracy of the histogram equalization procedure as well as the usefulness of the coprocessor.

In summary, this project entailed the effective creation of an image enhancement coprocessor employing FPGA technology. Verilog and C were used to create efficient hardware and software, while Vivado was used to expedite integration. The comparison of MATLAB and Vivado results proved the accuracy of the histogram equalization procedure as well as the usefulness of the coprocessor.

1.2. Problem Statement:

Image enhancement is a critical step in many applications, including medical imaging, surveillance systems, and digital photography. Traditional image enhancement methods, such as contrast stretching and equalization, have limits and downsides that impair the treated image's quality and accuracy. Implementing histogram equalization on a general-purpose CPU may be

computationally costly and sluggish, affecting the application's real-time performance. Although FPGA technology is an economical choice for image processing applications, creating and implementing a coprocessor for histogram equalization may be complicated and difficult.

As a consequence, the project intends to create an FPGA-based image enhancement coprocessor[2] [3] that uses histogram equalization to give efficient and accurate results. In order to conduct histogram equalization on the image in a timely and effective way, the coprocessor must create correct histograms. The hardware and software components will be written in Verilog and C, respectively, with integration taking place in Vivado.

Testing and comparison with MATLAB will be performed to confirm accuracy and efficacy of the coprocessor. The project's purpose is to offer a simplified solution for image processing applications by overcoming the constraints of standard image enhancing approaches.

The creation of this coprocessor has the potential to improve a variety of sectors, including medical imaging, surveillance systems, and digital photography. This initiative may help to progress these disciplines and enhance their applications by delivering more efficient and accurate image processing algorithms.

1.3. Approach

The project entitled "Co-Processor Design for Image Enhancement using FPGA" aimed to design a coprocessor for image enhancement using FPGA. The approach involved several steps, including:

1.3.1. Algorithm Design:

The first step was to design an algorithm for histogram equalization without using built-in functions in MATLAB.

1.3.2. Fixed-Point Conversion:

Once the algorithm was designed, the fixed-point conversion Q16.16 was done in

MATLAB.

1.3.3. Verilog and C Integration:

The next step was to develop the verilog and C parts in Vivado.

1.3.4. FPGA Board Selection:

We used the Arty Z7-10 Zynq 7000 board for implementation.

1.3.5. Integration:

We integrated all the components to create the coprocessor for image enhancement.

1.3.6. Testing:

Finally, we compared the result of the coprocessor with the results obtained from MATLAB to validate its accuracy.

Using FPGA technology, we were able to construct an efficient coprocessor for image enhancement. The coprocessor might be utilized in a variety of real-time image processing applications, including medical imaging, surveillance systems, and video processing.

1.4. Objectives:

- The purpose of this project is to provide a hardware-based solution for histogram equalization utilizing FPGA boards and to accomplish the following objectives:
- To attain great precision in the image enhancing process.
- Optimize the processing time of image enhancing algorithm.
- To assure resilience and dependability of the coprocessor design.
- To investigate possible uses the coprocessor in real-time image enhancing systems.
- To provide the groundwork for future development and research in image enhancement utilizing FPGA technology.

1.5. Specifications:

The Arty Z7-10 ZYNQ 7000™ All Programmable System-on-Chip (APSoC) utilized in this research, is a powerful and adaptable platform. Both Programmable Logic (PL) and Processing

System (PS) components are included in the SoC [4]. The PL component of the SoC is highly programmable, enabling for the implementation of custom logic architectures. The PL is used in this project to compute the histogram of the input image, which is required for the histogram equalization procedure. The PS is used to perform the actual histogram equalization of the image using the histogram calculated by the PL. Because the SoC uses both the PL and PS components, effective parallel processing is possible, resulting in quicker performance and better throughput. Overall, the Arty Z7-10 ZYNQ 7000 is a great platform for this project, giving the capabilities and flexibility needed for effective histogram equalization implementation.

1.6. Deliverables:

- This project's deliverables include: a functioning model/prototype of an image processing system capable of performing histogram equalization on images.
- Verilog code are used for histogram computation, while C code are used for fixed-point conversion.
- Verilog and C code integration with the Arty Z7-10 ZYNQ 7000 APSoC platform.
- Testing the system to check that it appropriately performs histogram equalization.
- Documentation of the design, implementation, and testing processes for the system.

1.7. Organization of Thesis:

This thesis is structured as follows:

Chapter 1: Introduction

This chapter offers a summary of the project, including its goals and problem description. It also provides a quick overview of the Arty Z7-10 ZYNQ 7000, the hardware utilized in this project. It also emphasizes the approach employed in the project development process.

Chapter 2: Review of Literature

The literature on image processing, histogram equalization, and FPGA-based solutions is

reviewed in this chapter. It contains a thorough examination of the state-of-the-art in image processing utilizing FPGAs.

Chapter 3: System Design

This chapter discusses the system's overall design, including the hardware and software components. It addresses design concerns such as system architecture, block diagrams, and component functioning. This chapter also provides implementation details, such as Verilog coding for histogram computation and equalization, fixed-point conversion, and the integration of hardware and software components.

Chapter 4 : Results and Analysis

This chapter assesses the system's performance using several measures such as processing time, resource use, and power consumption. It also includes a comparison of the findings to current literature.

Chapter 5: Future Projects

This chapter describes possible future work that might be done to improve the system. It addresses potential enhancements to the hardware and software components, as well as potential new functionality.

Chapter 6 : Conclusion:

This chapter outlines the project's research activity and emphasizes the main accomplishments. It finishes with a description of the project's contributions to image processing as well as the FPGA-based implementations.

Appendices: These provide more information regarding the system design, Verilog codes, and results. They also provide a list of references that were utilized in the project.

CHAPTER # 02

LITERATURE REVIEW

2.1. BACKGROUND

2.2. INTRODUCTION TO FPGA

2.2.1. ARTY Z7-10

2.3. FPGA DESIGN AND PROGRAMMING

2.4. IMAGE PROCESSING

2.4.1. IMAGE ENHANCEMENT

2.5. HISTOGRAM EQUALIZATION

2.5.1. METHODOLOGY

2.5.2. MATHEMATICAL CONCEPT

2.5.3. IMPACT OF HISTOGRAM EQUALIZATION ON FINAL
OUTPUT

2.5.4. PSEUDO CODE IMPLEMENTATION

2.6. EXISTING LITERATURE RELATED TO THE TOPIC

2.7. PROBLEM FORMULATION OF THE TOPIC

Literature Review

The need for high-performance and efficient computer systems is increasing at an unprecedented pace in today's digital world. Because of their reconfigurability and high parallelism, Field Programmable Gate Arrays (FPGAs) have emerged as a popular alternative for implementing complicated algorithms in hardware. Histogram Equalization is a popular method for improving image contrast. However, because of their high computational cost, classic histogram equalization techniques may not be suited for real-time processing. As a result, FPGA-based solutions of histogram equalization have received a lot of attention in recent years.

2.1. Background

This chapter introduces the Arty Z7-10 ZYNQ 7000 [4] and the notion of histogram equalization. It also contains a survey of the literature on the relevant topics to the project. To support and enhance the goal of this work, each of the subjects will be covered briefly. The goal of this chapter is to provide an overview of the project hardware and software components as well as current research in this field.

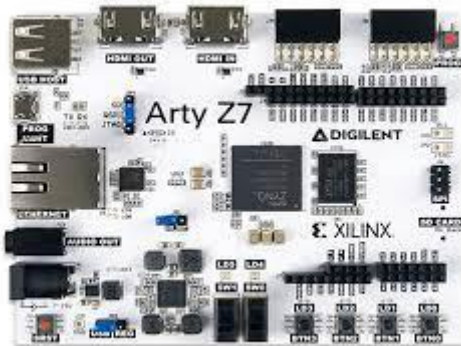


Figure 1 : Arty Z7

2.2. Introduction to FPGA

FPGAs are integrated circuits that may be programmed to construct custom logic design. Unlike typical ASICs, FPGAs may be reprogrammed even after they have been created, making them versatile and adaptable to a broad variety of applications. As a result, they are widely used in a

variety of areas, including telecommunications, aircraft, and digital signal processing. FPGAs combine the greatest design qualities of ASICs [5] and processor-based systems, which has led to their widespread usage across many sectors. FPGAs provide hardware-timed performance and dependability. The versatility of reprogrammable silicon is similar to that of software operating on a processor-based system, but it is not limited by the number of processing cores available. FPGAs are distinguished from CPUs by their real parallel nature. Because there is no scarcity of resources, various processing activities do not have to compete for the same ones. Every separate job is carried out by a specialized area of the silicon chip and may operate autonomously and independently of other CLBs. Consequently, the performance of one portion of the application has no effect on the other operations.

2.2.1. Arty Z7-10:

The Arty Z7-10 development board is built on the Xilinx Zynq-7000 System on Chip (SoC) [6]. This SoC combines a dual-core ARM Cortex-A9 [7] CPU with programmable logic, giving it a flexible platform for co-designing hardware and software. The device's programmable logic is constructed using Xilinx's Artix-7 FPGA, which contains 17,600 logic cells and 28,000 flip-flops. The board also has peripherals like USB connections, Ethernet connectors, and HDMI output for connecting with other devices.



Figure 2: ARM core Processor

For embedded system design, digital signal processing, and FPGA prototyping, the Arty Z7-10 is a popular option. Because of its strong ARM processor and FPGA fabric, it is well suited for implementing sophisticated algorithms and specialized hardware designs. Furthermore, its extensive collection of I/O connections make it an ideal platform for interacting with other devices and sensors.

2.3. FPGA Design and Programming

A hardware description language (HDL) or a schematic design may be used to define the behavior of the FPGA. The HDL format is ideal for dealing with complicated structures because it enables the user to express them mathematically rather than drawing each component manually. Using a schematic entry, on the other hand, allows for easier visualization of the design.

Following that, an electronic design automation tool is used to create a technology-mapped net-list. This net-list is then changed to fit the real FPGA design via a procedure known as place-and-route, which is often performed using the FPGA developer's proprietary place-and-route software. The mapping, placement, and routing results are validated by the user using time analysis, simulation, and other verification methods. Following the completion of the design and validation processes, the FPGA is (re)configured using a binary file created by the FPGA developer's proprietary software. This file is sent to the FPGA using its programming connection.

An FPGA application developer does simulations at different phases of the design process in a typical design flow. The RTL [8] description in VHDL or Verilog is first emulated by constructing test benches to monitor the system's behavior and consequences. Following that, when the synthesis engine maps the design to a net-list, the net-list is transformed to a gate level description, and simulations are run to ensure error-free synthesis. Finally, the design is implemented on the FPGA, enabling propagation delays to be added, and the simulation is performed again with these values annotated into the net-list.

2.4. Image Processing

The alteration of photos to improve their quality or extract relevant information is known as image processing. Filtering, segmentation, and feature extraction may all be used to change an image to make it more usable for a certain purpose.

2.4.1. Image Enhancement:

Image enhancement is a subset of image processing that focuses on improving the visual quality of an image. This may be performed by increasing contrast, sharpening the image, eliminating noise or blurring, and using other techniques to make the image more visually pleasing or easier to comprehend. The goal of image enhancement is to improve the interpretability or quality of the image while retaining the integrity of the original data.

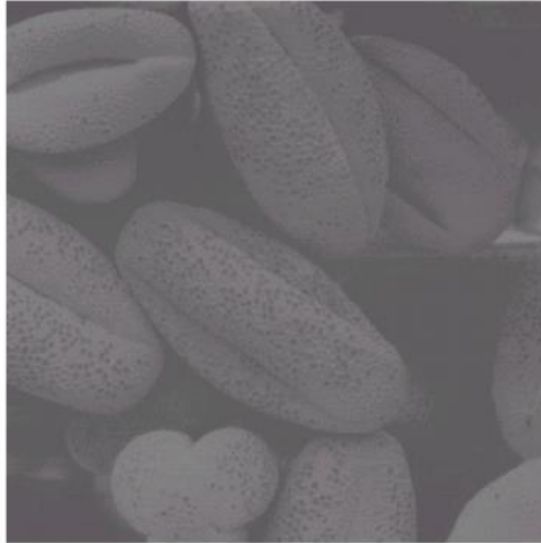


Figure 3 : Original Image



Figure 4 : Enhanced Image

Image processing methods include point processing, neighborhood processing, and global processing.

Point processing [9] entails performing actions on each pixel of an image independently of its neighbors. It implies that a pixel output value is decided exclusively by its own input value and a mathematical function. Point processing methods include, for example, brightness modification, contrast adjustment, and thresholding.

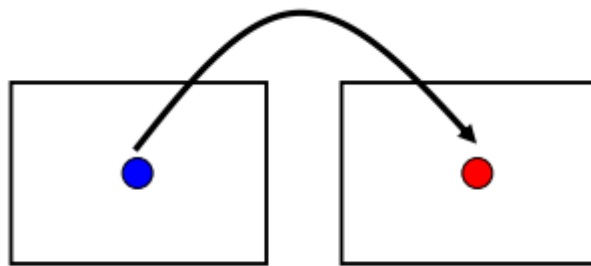


Figure 5 : Point Processing

Neighborhood processing [10], also known as local processing, is the process of calculating a new pixel value depending on the values of the pixels around it. The output value of a pixel in this approach is determined not only by its own input value but also by the values of its nearby pixels. Neighborhood processing methods include filtering, which includes smoothing, sharpening, and edge detection.

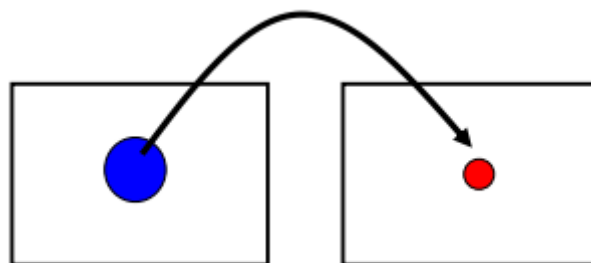


Figure 6 : Neighborhood Processing

Global processing [11] entails processing the whole image in its entirety. A pixel output value is determined by the values of all the pixels in the image. Image compression, image segmentation, and feature extraction are examples of global processing methods. Global processing is generally employed for high-level image analysis and interpretation since it is more computationally costly than point and neighborhood processing.

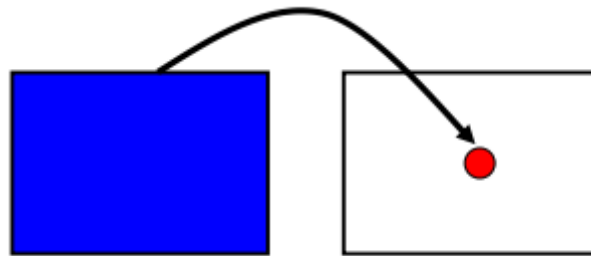
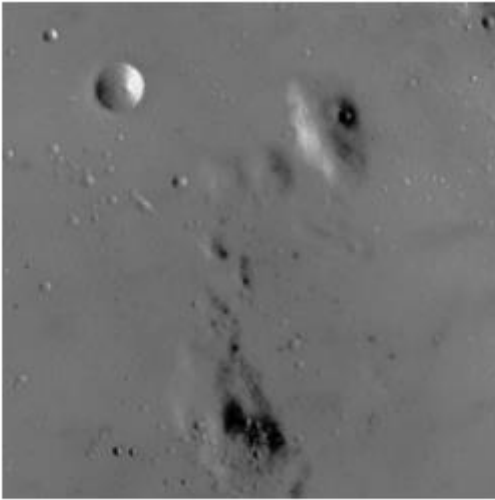


Figure 7 : Global Processing

2.5. Histogram Equalization

Histogram Equalization [12] is a common image processing method for improving image contrast. The approach works by dispersing the pixel values of an image to create a more uniform histogram, which results in a more visually appealing image. In histogram equalization, the transformation function is a monotonically growing function that translates the input pixel values to the output values. Histogram equalization is a global image processing method that modifies the pixel values of a whole image to get the desired outcome. Neighborhood processing approaches, on the other hand, include changing the pixel values of a specific pixel depending on the values of its nearby pixels.



Original Image

Figure 8 : Original Image



Histogram Equalization

Figure 9 : Histogram Image

There are many variants of the fundamental histogram equalization approach, including contrast limited adaptive histogram equalization (CLAHE) and adaptive histogram equalization (AHE). CLAHE improves simple histogram equalization by restricting the highest pixel value in each

histogram bin, reducing noise over-amplification in low-contrast portions of the image. AHE adopts a different technique, calculating the histogram [13] for tiny portions of the image, resulting in adaptive contrast enhancement that is more suitable for photos with non-uniform illumination. There are many variants of the fundamental histogram equalization approach, including contrast limited adaptive histogram equalization (CLAHE) and adaptive histogram equalization (AHE). CLAHE improves simple histogram equalization by restricting the highest pixel value in each histogram bin, reducing noise over-amplification in low-contrast portions of the image. AHE adopts a different technique, calculating the histogram [13] for tiny portions of the image, resulting in adaptive contrast enhancement that is more suitable for photos with non-uniform illumination.

2.5.1. Methodology

1. The formula for histogram equalization is quite simple and consists of the following steps:
2. Determine the inputs: The input for histogram equalization is an image.
3. Calculate the histogram: Make a histogram of the supplied image. This is the number of pixels in the image that have each intensity level.
4. Calculate the cumulative distribution function (CDF): Calculate the histogram's CDF [14]. This is a function that translates each intensity level to the likelihood that a pixel in the image has an intensity level that is less than or equal to that intensity level.
5. Calculate the equalized intensity values as follows: Using the CDF [15], compute the equalized intensity values for each pixel in the image.

6. Produce the equalized image: Create an image with equalized intensity values.

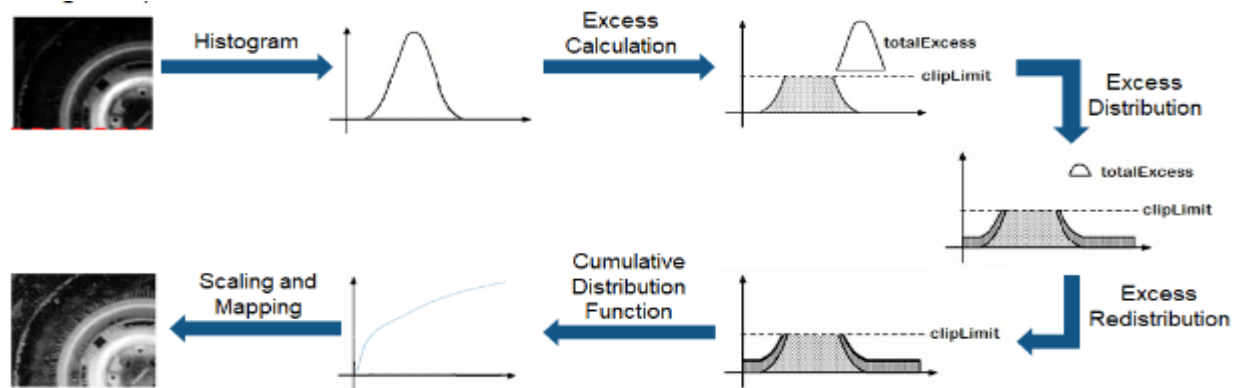


Figure 10 : MATLAB Flow Chart

2.5.2. Mathematical concept

The equation used for histogram equalization is given below:

$$s = T(r) = (L-1) * \sum P(r_k)$$

where r is the input image intensity, s is the output intensity, L is the number of possible intensity levels, and $P(r_k)$ is the probability of occurrence of the input intensity r_k . The function $T(r)$ maps the input intensity to the output intensity.

2.5.3. Impact of Histogram Equalization on Final Output:

The final outcome of the histogram equalization method is heavily influenced by the image intensity distribution. If the intensity distribution of an image is substantially skewed towards a certain intensity value, the equalization procedure will result in the majority of pixels having the same intensity value. This may result in the loss of visual details, particularly in regions with low contrast in the original image.

If, on the other hand, the intensity distribution of the image is uniform, the histogram equalization procedure will result in a more balanced image with higher contrast and visual attractiveness.

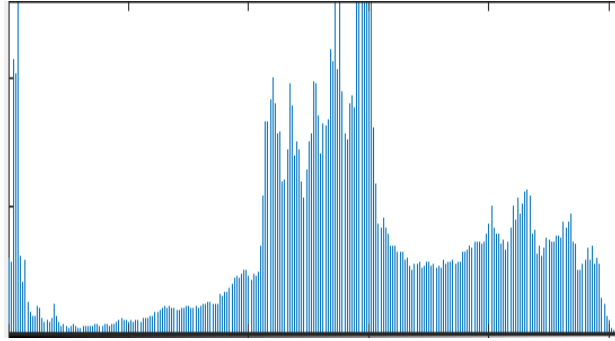


Figure 11 : Original Image Histogram

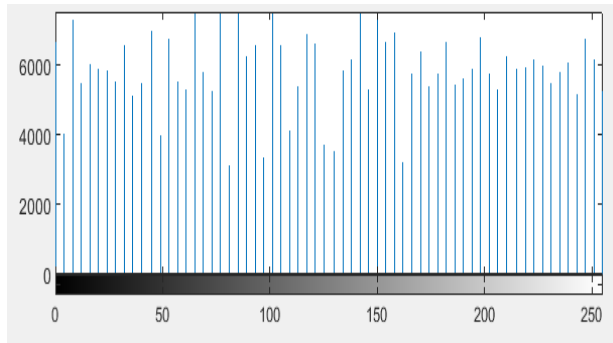


Figure 12 : Equalized Image Histogram

2.5.4. Pseudo code implementation:

```
// Assuming input image is in grayscale format with pixel values ranging from 0 to 255

// Step 1: Compute the histogram of the image
histogram = [0]*256 // Initialize the histogram to zero for each intensity level
for each pixel in the image:
    intensity = pixel.value
    histogram[intensity] += 1

// Step 2: Compute the cumulative distribution function (CDF) of the histogram
cdf = [0]*256 // Initialize the CDF to zero for each intensity level
sum = 0
for i in range(256):
    sum += histogram[i]
    cdf[i] = sum

// Step 3: Compute the normalized CDF
total_pixels = image.width * image.height
norm_cdf = [0]*256 // Initialize the normalized CDF to zero for each intensity level
for i in range(256):
    norm_cdf[i] = (cdf[i] / total_pixels) * 255

// Step 4: Map each pixel value to its corresponding normalized CDF value
for each pixel in the image:
    intensity = pixel.value
    pixel.value = norm_cdf[intensity]

// Output the equalized image
output_image = image
```

Figure 13 : Pseudo Code

2.6. Existing Literature Related to the topic:

The work 'FPGA-Based Histogram Equalization for Image Processing' by Nahin Ul Sahad, Afsana Afrin, and Md Nazrul Islam Mondal from Rajshahi University of Engineering and Technology is one example of existing literature on this issue. The authors used VHDL to develop histogram equalization on a Xilinx Artix-7 FPGA in this research. According to the

findings, the FPGA-based system was 1.5 thousand times quicker than typical CPU-based systems.

The work 'FPGA Implementation of Image Improvement Techniques' by Atul Srivastava discussed the usage of numerous techniques for image enhancement on a Virtex 4 FPGA, including Power Law, Laplacian, and Sobel Filter. The Sobel filter was shown to yield erroneous edge detection, but the Laplacian filter took longer. The Power Law approach has a good correlation value but a lower success rate when it came to contrast augmentation. The research presents an informative analysis of the benefits and limitations of various image enhancing approaches on FPGA.

2.7. Problem Formulation of the Topic:

Traditional CPU-based histogram equalization techniques, on the other hand, have restricted performance because of the high computational needs. To address this restriction, we suggest utilizing the Arty Z7-10 FPGA to construct a hardware-based histogram equalization technique.

The primary goal of this project is to develop and construct a hardware architecture capable of performing image histogram equalization. To achieve high-speed performance, the suggested system will take use of FPGA's flexibility and parallel processing capabilities. In order to reduce hardware expenses, the system will also be tuned for resource consumption. The suggested system will be tested utilizing several image kinds with variable features. The system's performance will be compared to that of typical CPU-based histogram equalization techniques.

CHAPTER # 03

DESIGN AND DEVELOPMENT

3.1. DESIGN FLOW

3.2. HARDWARE AND SOFTWARE PART

3.3. SOFTWARE PART

3.3.1. DESIGNING ALGORITHM

3.3.2. HEX FILE GENERATION USING MATLAB

3.4. HARDWARE DESIGNING USING HDL

3.4.1. BRAM MODULE

3.4.2. HISTOGRAM MODULE

3.5. SOFTWARE PART USING C

3.6. SOFTWARE / HARDWARE INTEGRATION

3.7. HARDWARE INTERFACING

3.8. HARDWARE REQUIREMENT

3.9. SOFTWARE REQUIREMENT

Design and Development

The Arty Z7-10 ZYNQ-7000 is used in the project "Co-Processor Design for Image Enhancement Using FPGA."

The Arty Z7-10 board is powered via a micro-USB cable attached to a computer's USB port or a USB wall adapter. Alternatively, an external power source connected to the barrel jack on the board may be used to power the board.

A micro-USB cable may be used to make a USB-UART connection between the Arty Z7-10 and a computer. The board also has a USB OTG connector for programming the FPGA or developing a USB host or device application. Additionally, the board has Ethernet and HDMI ports for connecting to a network or a display device, respectively.

MATLAB will generate the hex file from the image. This file will then be saved in the Arty Z7-10 board's BRAM (Block Random Access Memory). The FPGA's Programmable Logic (PL) will access the BRAM and compute the image histogram. The Histogram IP will be created and coded in Vivado using Verilog.

When the Histogram IP has finished its work, the data is sent to the Processing System (PS) for fixed-point histogram stretching. In Vivado HLx, the histogram stretching will be coded in C. Vivado will be used to merge the several IPs.

Finally, a bitstream file containing all of the settings and programming information necessary to program the FPGA will be created. Using a programming cable, this bitstream file will be loaded into the Arty Z7-10 board, and the histogram equalization method will be implemented on the FPGA.

3.1. Design Flow:

System design, software-hardware partitioning, floating-point behavioral description, fixed-point conversion, RTL Verilog implementation, functional verification, synthesis, gate netlist generation, timing and functional verification, layout, software and hardware co-verification,

integration, and testing are typical stages in the design flow of an embedded system. Each phase in the design flow is explained in detail below:

1. **System Development:** The overall system requirements and specifications are determined during the system design process. This comprises determining the functionality, performance objectives, interfaces, and restrictions of the system. The system design acts as a roadmap for the succeeding design flow processes.
2. **Floating-Point Behavioral Description:** The system's behavior is described using floating-point arithmetic at this step. To capture the intended accuracy and precision of computations, algorithms and operations are expressed using floating-point representations. This behavioral description serves as the foundation for later software and hardware implementations.
3. **Software-Hardware Partitioning:** The functionality is partitioned across software and hardware components based on the system architecture and performance requirements. Tasks suitable for software implementation are selected, while those demanding high performance or specific hardware acceleration are assigned to hardware.
4. **Development of Software and Hardware:** The software and hardware components are built separately depending on partitioning choices. Writing code in a high-level programming language, such as C or C++, to implement the desired functionality is part of software development. The process of developing hardware modules utilizing RTL Verilog or other hardware description languages to express digital circuits is known as hardware development.
5. **Fixed-Point Conversion:** The floating-point behavioral description is converted to fixed-point form in this stage to enhance hardware resource consumption and efficiency. The algorithms and procedures are changed to work with fixed-point data types, and the precision and scaling factors for fixed-point arithmetic are calculated.
6. **Implementation of RTL Verilog:** RTL descriptions in Verilog or other hardware description languages are used to implement the hardware components. RTL covers the structure and behavior of digital circuits, including data flow, control signals, and interconnections between modules. The RTL designs capture the intended functionality and are utilized in the verification and synthesis phases that follow.

7. Functional verification is conducted after RTL Verilog implementation to guarantee that the hardware and software components act appropriately in accordance with the system design and requirements. To ensure the system's functional accuracy, several verification approaches including as simulation, formal verification, and testbench creation are utilized.
8. Synthesis and Gate Netlist Generation: After functional verification, synthesis techniques are employed to turn RTL descriptions into gate-level representations. The synthesis process converts the RTL design into a gate netlist, which specifies the circuit in terms of gates, flip-flops, and interconnections.
9. Time and Functional Verification: Timing and functional verification are performed on the gate netlist to validate that the design fulfils the timing restrictions and functions as intended. Timing analysis is used to ensure that the circuit fulfils the needed timing parameters, while functional verification validates the gate-level design against the original system requirements.
10. Layout: The gate-level design is physically executed at the layout step by positioning and routing the circuit components on the target chip or FPGA. The layout process guarantees that physical connection, signal integrity, and design standards and limitations are followed.
11. Hardware and software Co-verification: To guarantee optimal interaction and operation, the software and hardware components are integrated and co-verified. Through simulation or emulation, software is tested with hardware to validate the software-hardware interface and overall system behavior.
12. Integration and testing: The last stage is to integrate all of the software and hardware components into a fully functional embedded system. The integrated system is rigorously tested to ensure that it functions properly, performs well, and meets the system requirements. Unit testing and integration testing are examples of testing.

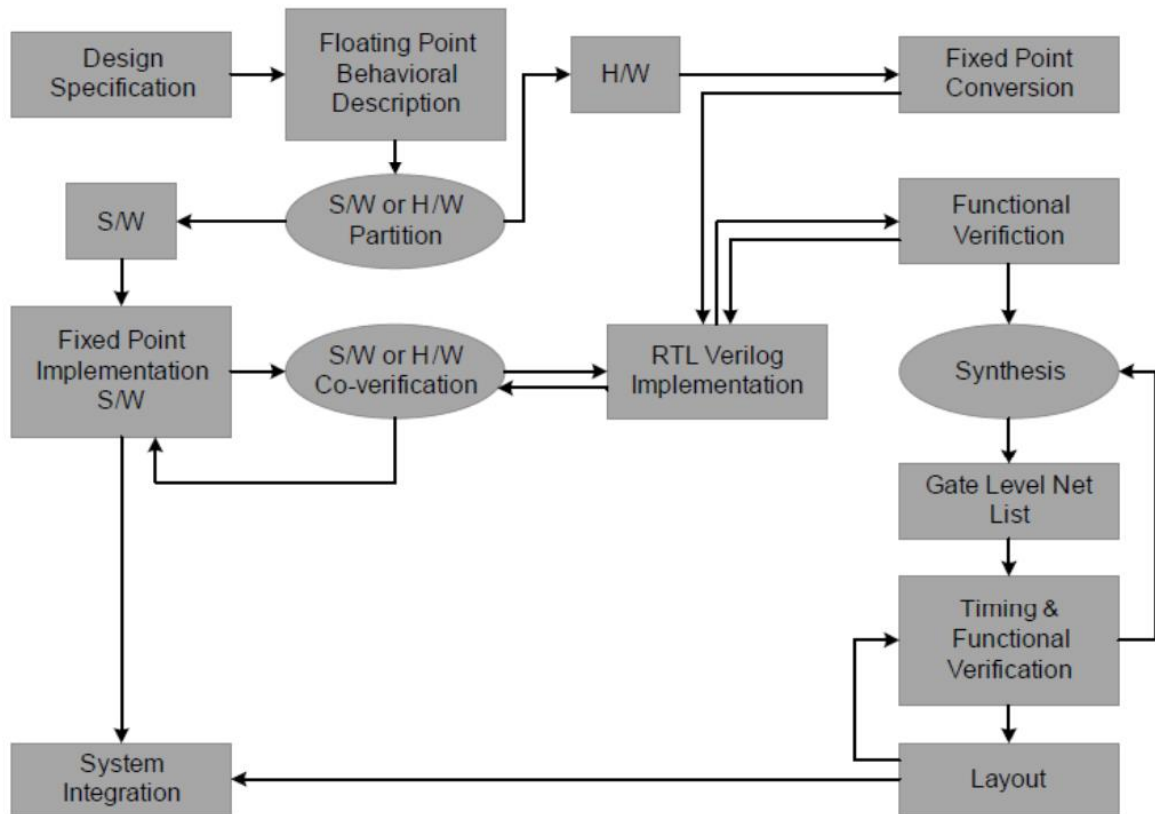


Figure 14 : Project Block Diagram

3.2. Hardware and Software part:

This project includes both software and hardware components. In the software section, we used MATLAB to create the method for histogram equalization, leveraging Q16.16 fixed point conversion for precise data representation. Following that, the hex file generated from the image will be utilized in the hardware component. The histogram module will be developed in Verilog, while the fixed-point histogram stretching will be done in C in Vivado HLx. To assess the mean square error and speed, the final output from Arty Z7-10 will be compared to the MATLAB output acquired from the first software phase. The project success will be determined by the accuracy and efficiency of the developed histogram equalization technique.

3.3. Software part:

The software part comprises of coding of MATLAB and C.

3.3.1. Designing Algorithm

MATLAB is a high-level programming language and interactive environment that is extensively used in mathematics, graphics, and algorithm development. In this research, the histogram equalization technique was initially built and evaluated in MATLAB.

The goal of constructing the algorithm in MATLAB was to test and evaluate it before putting it on hardware using an FPGA. The technique may be simply built and tested on example photos in MATLAB to ensure that it is running properly and generating the intended results. The following stages are included in the algorithm design for histogram equalization:

1. Image acquisition : The initial step is to get the input image.
2. Grayscale conversion: The RGB-to-grayscale conversion formula is used to convert the input image to grayscale.
3. Histogram: The grayscale image's histogram is computed using a fixed-point conversion of Q16.16.
4. Probability Density Function (PDF): The histogram's PDF [16] is computed by dividing each histogram value by the total number of pixels in the image.
5. Cumulative Distribution Function (CDF): The cumulative distribution function (CDF) of the PDF is computed by adding the PDF values from the first bin to the current bin.
6. Input intensity to output intensity mapping: The CDF values are used to transfer the input intensity values to the output intensity values.
7. Equalized histogram calculation: The equalized histogram is computed using the mapped output intensity values.
8. Conversion of the equalized histogram to an image: A fixed-point conversion of Q16.16 is used to convert the equalized histogram to an image.
9. Comparison of the output and input images: The mean square error (MSE) [17] value is used to compare the output image to the input image.
10. The input and output images are displayed: The input and output photos are shown for comparison.

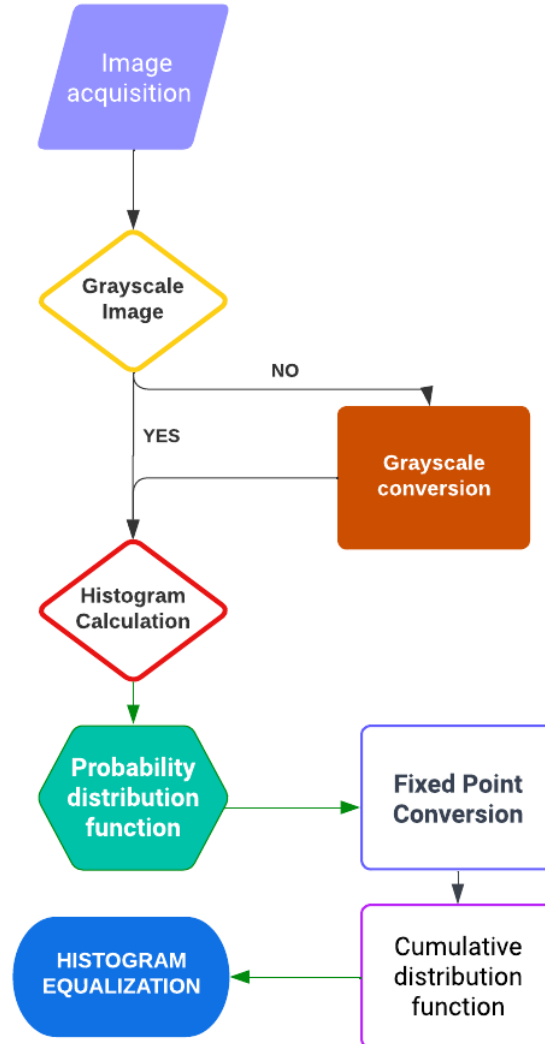


Figure 15 : MATLAB Algorithm

3.2.2. Hex File Generation using MATLAB:

To create a hex file, the pixel values of the input image are converted into binary format, with a certain amount of bits reserved for each pixel value. The binary data is subsequently transformed into ASCII code, a character encoding method in which each digital digit is represented as a character [18].

The ASCII code is then saved in a file with an extension “.hex”. This file provides the image binary data in a format that hardware devices like FPGAs may readily read.

3.4. Hardware designing using HDL

Hardware designing using Verilog for this project involves creating two modules that will be implemented in the Programmable Logic (PL) of the Arty Z7-10 FPGA board.

3.4.1. BRAM module

Once created in MATLAB, the hex file will be saved in the Arty Z7 FPGA board's BRAM module. BRAM (Block RAM) [19] is a type of memory that is utilized in FPGAs and is especially intended for fast access and efficient use of FPGA resources. BRAM is useful for storing data that must be retrieved often and fast, and will be used to store image data in the FPGA's PL. The BRAM will be attached to the FPGA's Input/Output (I/O) pins and will have a read and write interface. This module will be written in Verilog code and will implement the BRAM's read and write capability.

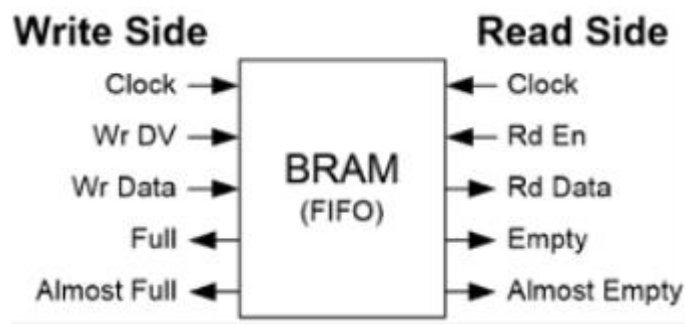


Figure 16: BRAM

3.4.2 Histogram Module

The histogram module in Verilog is in charge of computing an image's histogram. It takes pixel values from the BRAM module and counts the number of times each intensity value appears. The histogram presents a frequency distribution of pixel intensities, which is required by many image processing algorithms.

In Verilog, the histogram module is implemented by designing a combinational logic circuit that iterates over each pixel value. Based on the received pixel value, it employs a counter to increase the matching bin in the histogram array. The histogram module effectively analyses the image input and provides a histogram array that reflects the intensity frequency distribution. A finite state machine (FSM) and numerous crucial components are used in our design's

histogram module. These elements collaborate to provide an efficient histogram calculation:

Initialization Counter: A counter is used to aid in the process of initialization. It correctly records the quantity of pixels processed, guaranteeing optimal synchronization and operation sequencing.

This module creates the pixel addresses needed to access the image RAM. It guarantees that pixels are read from RAM in the right sequence, preserving data integrity.

Image RAM: The image RAM is a memory component that holds the pixel data of the input image. The pixel address generator generates the correct address for accessing the image RAM and retrieving the pixel values.

Multiplexer (MUX): The MUX [20] is in charge of choosing the input source between the address counter and the image RAM output during startup. The MUX selects the address counter generated during the startup phase to load the image pixels into the histogram RAM.

Histogram RAM: This memory component is essential for storing histogram values. The pixel address provided by the pixel address generator is used to access the histogram RAM, allowing the histogram to be calculated for each pixel value.

Our histogram module properly manages initialization, pixel reading, histogram calculation, and storage in the histogram RAM by combining these components and leveraging a well-designed finite state machine. The FSM [21] orchestrates the flow and scheduling of these activities, guaranteeing dependable and efficient image data processing.

Xilinx's Project Navigator 14.2 was used for all Verilog code, while ISim was used for simulation. The histogram module in Verilog is in charge of computing an image's histogram. It takes pixel values from the BRAM module and counts the number of times each intensity value appears. The histogram presents a frequency distribution of pixel intensities, which is required by many image processing algorithms.

In Verilog, the histogram module is implemented by designing a combinational logic circuit that iterates over each pixel value. Based on the received pixel value, it employs a counter to increase the matching bin in the histogram array. The histogram module effectively analyses the image input and provides a histogram array that reflects the intensity frequency distribution. A finite state machine (FSM) and numerous crucial components are used in our design's histogram module. These elements collaborate to provide an efficient histogram calculation:

Initialization Counter: A counter is used to aid in the process of initialization. It correctly records the quantity of pixels processed, guaranteeing optimal synchronization and operation sequencing.

This module creates the pixel addresses needed to access the image RAM. It guarantees that pixels are read from RAM in the right sequence, preserving data integrity.

Image RAM: The image RAM is a memory component that holds the pixel data of the input image. The pixel address generator generates the correct address for accessing the image RAM and retrieving the pixel values.

Multiplexer (MUX): The MUX [20] oversees choosing the input source between the address counter and the image RAM output during startup. The MUX selects the address counter generated during the startup phase to load the image pixels into the histogram RAM.

Histogram RAM: This memory component is essential for storing histogram values. The pixel address provided by the pixel address generator is used to access the histogram RAM, allowing the histogram to be calculated for each pixel value.

Our histogram module properly manages initialization, pixel reading, histogram calculation, and storage in the histogram RAM by combining these components and leveraging a well-designed finite state machine. The FSM [21] orchestrates the flow and scheduling of these activities, guaranteeing dependable and efficient image data processing.

Xilinx's Project Navigator 14.2 was used for all Verilog code, while ISim was used for simulation.

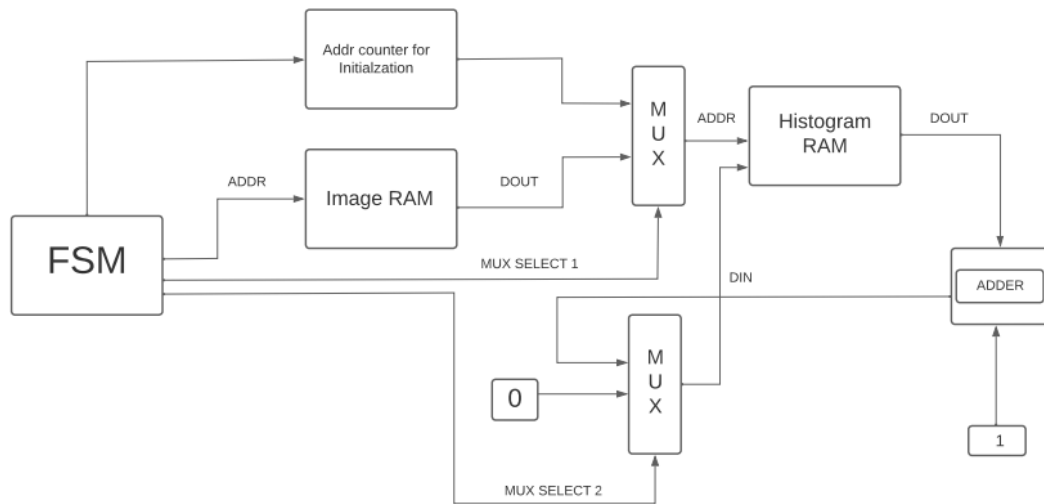


Figure 17 : Histogram Block Diagram

3.5. Software part using C:

The fixed-point histogram stretching approach was used in the software section built in C in Vivado HLx. This procedure was created primarily to boost the image's contrast and visual quality. Here's an outline of how the software portion did it:

Software Development: The fixed-point histogram stretching approach was implemented in C code. This method acts on the image data's fixed-point representation acquired in the preceding fixed-point conversion phase. The code takes the image as input and does the required computations and adjustments to extend the histogram.

Calculation of the Histogram: The C code initially computes the histogram of the input image. This entails counting the number of times each pixel value appears in the image. The histogram represents the pixel distribution statistically, which is used to compute the intensity transformation necessary for stretching.

Histogram Stretching: The C code uses the stretching operation to alter the intensity values of the pixels in the image based on the histogram. Stretching redistributes the intensity levels throughout the full dynamic range, resulting in increased contrast and visual appeal.

Fixed-Point Arithmetic: Fixed-point arithmetic rules are used throughout the software implementation to execute calculations on the image data's fixed-point format. These criteria provide accurate and consistent calculations while taking the fixed-point format selected during the fixed-point conversion stage into consideration.

The software implementation on the PS (Processing System) component of the FPGA allows effective fixed-point data processing and usage.

3.6. Software/Hardware integration:

Two important IP cores were built in the project: the histogram module in Verilog and the histogram equalization module in C. The image histogram was calculated using the Verilog-based histogram module, and the equalization was conducted by the C-based histogram equalization module.

Vivado was used to incorporate these IP cores into the system. In Vivado, a block design representing the entire system architecture was constructed, which includes the Zynq Processing System and the IP cores.

The initial stage was to instantiate the Verilog-based histogram IP core inside the block architecture and link it to other system components. This includes specifying the histogram module's input and output ports and ensuring suitable linkages.

Similarly, the IP core for histogram equalization, implemented in C, was included into the block architecture. Its input and output ports were linked to the system's appropriate signals and interfaces.

Throughout the integration process, special care was taken to ensure that interfaces and connections between the IP cores and the Zynq Processing System were properly established. Input and output mappings were established to ensure proper pin assignments for FPGA connectivity.

In addition, system I/O interfaces were setup and, if necessary, linked to external devices or peripherals.

After the connections and settings were established, the architecture was validated to identify any timing violations or performance difficulties. Simulations were used to ensure that the integrated system worked properly.

After that, the design was synthesized and implemented in Vivado to create the bitstream file. This file included the FPGA's setup data, enabling it to perform histogram equalization on actual image data.

3.7. Hardware Interfacing:

The PC-to-Arty Z7 interface was used in this project to load the created bitstream file onto the Arty Z7 FPGA board. To establish the link, the programming wire was connected between the PC and the Arty Z7 board. On the PC, programming tools such as Xilinx Vivado were used to begin the programming process. The bitstream file holding the histogram equalization method configuration data was chosen and sent to the Arty Z7 board via the programming connection. Once coded, the Arty Z7 board's FPGA could run the histogram equalization method without the need for further PC interface.

3.8. Hardware Requirement:

The hardware requirements for this project include the following:

1. Arty Z7-10 board
2. Programming Cable
3. Power Adapter
4. PC/Laptop

3.9. Software Requirement:

The software requirement for this project includes the following:

1. MATLAB
2. Xilinx ISE
3. Xilinx Vivado

4. SDK
5. Eclipse

CHAPTER # 04

RESULT ANALYSIS

4.1. ALGORITHM DESIGNING ON MATLAB

4.4.1. HEX FILE GENERATION

4.2. HARDWARE RESULT ANALYSIS: VERILOG IMPLEMENTATION

4.2.1. BLOCK RAM MODULE

4.2.2. HISTOGRAM MODULE

4.3. SOFTWARE RESULT ANALYSIS: C IMPLEMENTATION

4.3.1. HISTOGRAM IMPLEMENTATION

4.3.2. HISTOGRAM EQUALIZATION

4.4. IP CORE INTEGRATION

4.4.1. BITSTREAM GENERATION

4.5. HARDWARE INTERFACING

Result Analysis

This section of the report focuses on the analysis and evaluation of the key project tasks discussed earlier. The purpose is to assess the effectiveness and performance of the implemented algorithms and techniques.

4.1. Algorithm Design on MATLAB

The algorithm implementation of image histogram equalization and fixed-point conversion starts with reading a grayscale image and converting it if required. To provide accurate intensity readings, the pixel values are clipped to the required range of $[0, 255]$. The image is then transformed to Q16.16 fixed-point format by scaling and rounding the values. The image's histogram is computed by counting the occurrences of each pixel value. The cumulative distribution function (CDF) is calculated, which represents the likelihood of each pixel value. Based on the CDF, a transformation function is developed that maps the original pixel values to new values for equalization. The image is changed by replacing each pixel value with its corresponding modified value. For display, the image is subsequently transformed back to 8-bit format. For visual examination, the original and equalized images, as well as their corresponding histograms, are plotted.

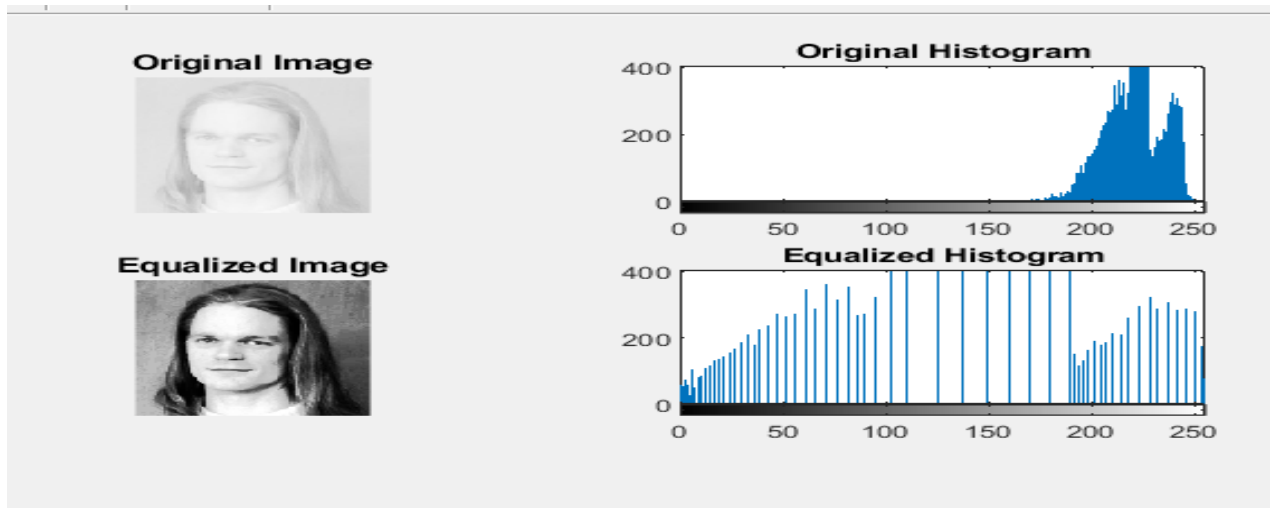


Figure 18 : Histogram Display

The developed approach's accuracy is then assessed by comparing the results to those obtained using built-in functions. The comparison reveals that the developed algorithm's accuracy is noticeably high, suggesting its usefulness in accomplishing histogram equalization and conducting fixed-point conversion.

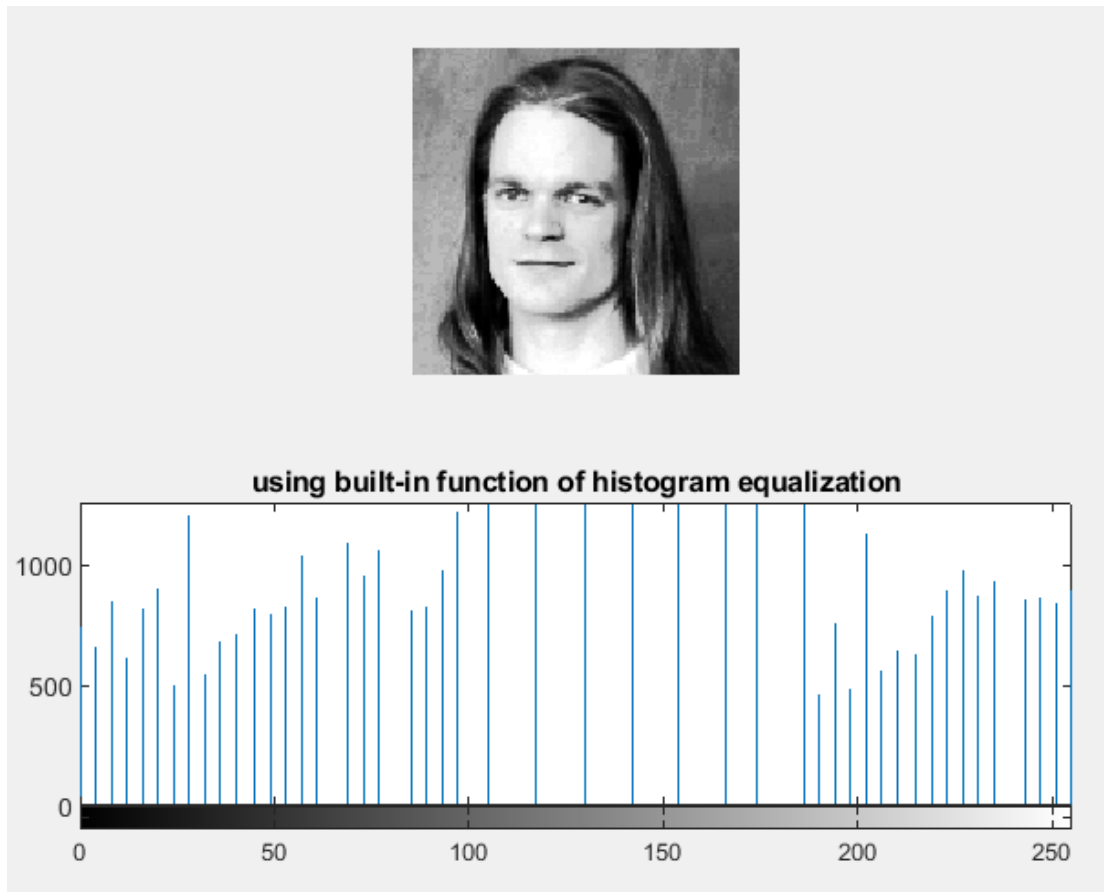


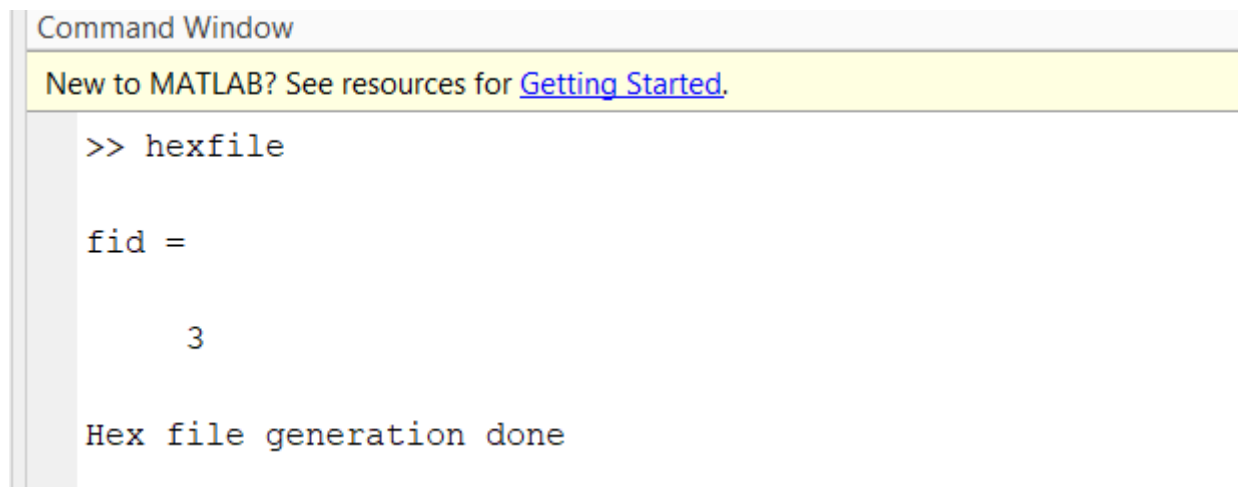
Figure 19 : Histogram Equalization using Built-in function

The MATLAB analysis indicated that image histogram equalization and fixed-point conversion methods were successfully implemented. Building upon this foundation, the project then moves on to the Verilog hardware design phase.

4.1.1. Hex file generation:

MATLAB was used to generate the hex file from the image pixels. The image pixels were processed and transformed to hexadecimal format, allowing for a more compact representation. This hex file arranges the image pixels so that they may be read and used in later phases of the project. The hex file, which serves as an intermediate representation of image data, improves data transmission, and is used in hardware modules such as the BRAM and histogram computation.

After finishing the hex file creation, the MATLAB command window shows a confirmation message indicating that the procedure has been completed. The notification indicates that "Hex file generation done."

A screenshot of the MATLAB Command Window. The window title is "Command Window". At the top, there is a yellow banner with the text "New to MATLAB? See resources for [Getting Started.](#)". Below the banner, the command prompt shows the execution of the 'hexfile' command. The output indicates that a file handle 'fid' has been assigned the value 3, and a confirmation message "Hex file generation done" is displayed.

```
Command Window
New to MATLAB? See resources for Getting Started.
>> hexfile

fid =

     3

Hex file generation done
```

Figure 20 : Command Window

The hex file output is presented below, providing a visual representation of the generated hex file.

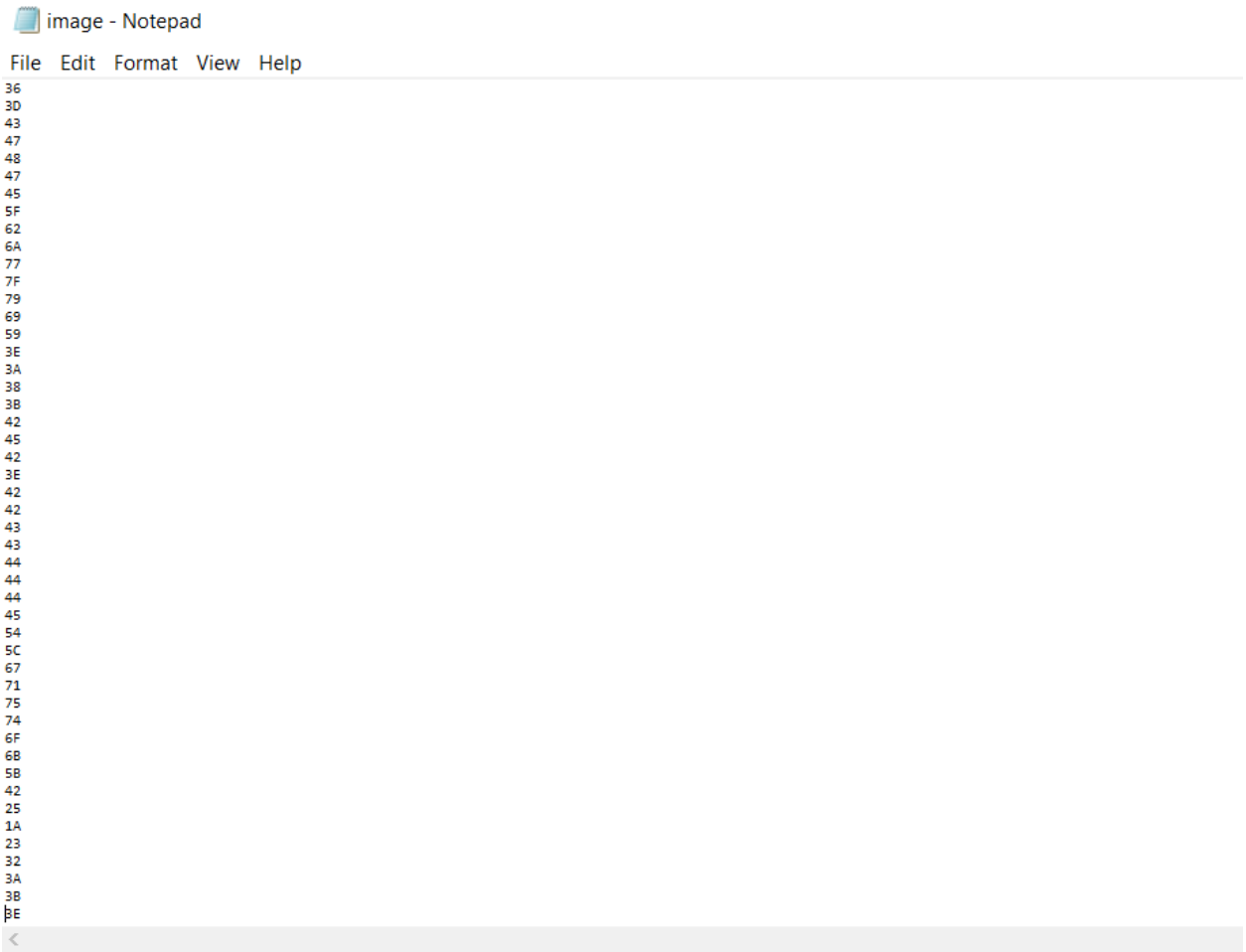


Figure 21: Hex File

4.2. Hardware Result Analysis: Verilog Implementation

The project consists of the development of two modules: BRAM (Block RAM) and Histogram. This section provides an in-depth examination of these modules, with an emphasis on their performance and usefulness.

4.2.1. Block RAM module

In this architecture, the BRAM module consists of a register array designated "mem" with a capacity of 4096 entries, each of which may hold 8 bits of data. The address bus, which controls which memory location to read or write to, is 8 bits long, allowing for a total of 256 distinct addresses. The 8-bit width of the BRAM assures that just one byte may be stored in each memory location.

A counter is used to effectively address the BRAM and retrieve data. The counter is 12 bits long, with a maximum count of 4095. The BRAM may be accessed sequentially or non-sequentially by incrementing the counter, depending on the required memory address. This addressing scheme guarantees that data from the BRAM module is stored and retrieved efficiently.

In this project, two types of design checks were used: software testing and hardware testing. A specific test bench with specified inputs incorporated in the code was created for software testing. After that, the test bench was emulated using Xilinx's ISim simulator. Figure 22 displays screenshots of the testing findings, demonstrating the software testing process and its consequences.

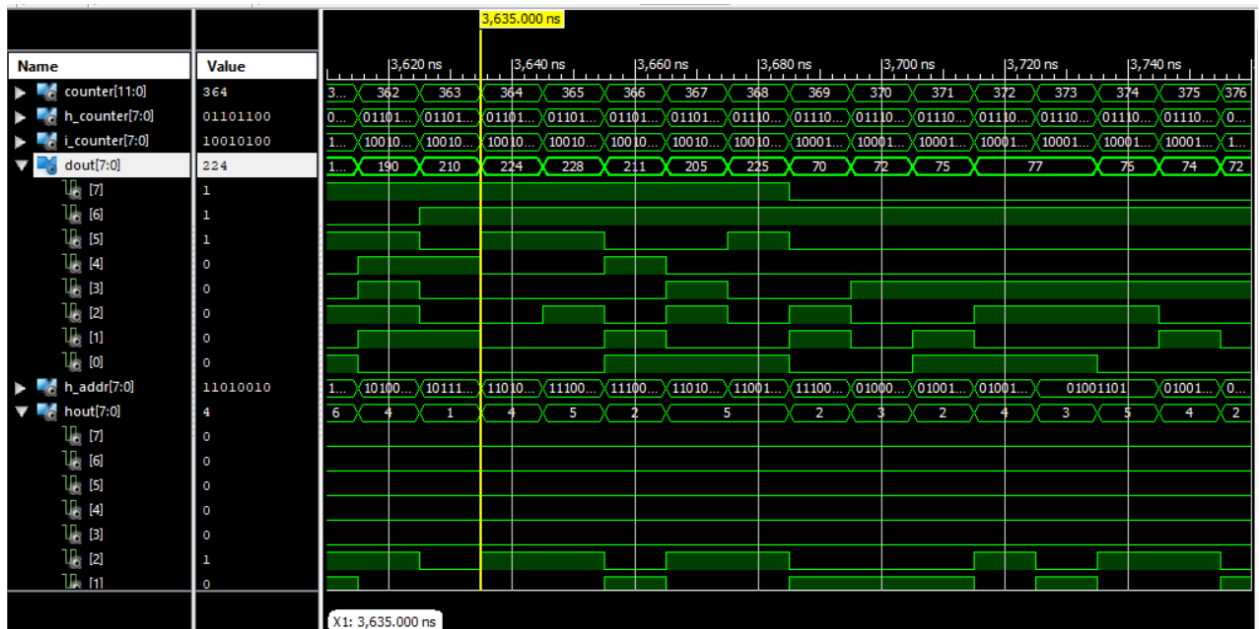


Figure 22 : BRAM simulation

The simulator creates a timing diagram that depicts the BRAM module's functioning. This module basically saves the image's pixel values retrieved from the hex file. The timing diagram depicts the signals and their transitions inside the BRAM module, allowing for a better understanding of its operation and how it saves image data.

Figure 23 shows the simulator's console window, which displays the output linked to the BRAM module. The terminal window displays critical information about the BRAM's functionality, such as status messages, debug output, and other pertinent facts. Analyzing the

console window helps in understanding the BRAM module's behavior and performance during simulation.

```
Console
ISim M.53d (signature 0x7dea747)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim> |
```

Figure 23 : Console Window

The simulator's console window is seen above, displaying the output linked to the BRAM module. The terminal window displays critical information about the BRAM's functionality, such as status messages, debug output, and other pertinent facts. Analyzing the console window helps in understanding the BRAM module's behavior and performance during simulation.

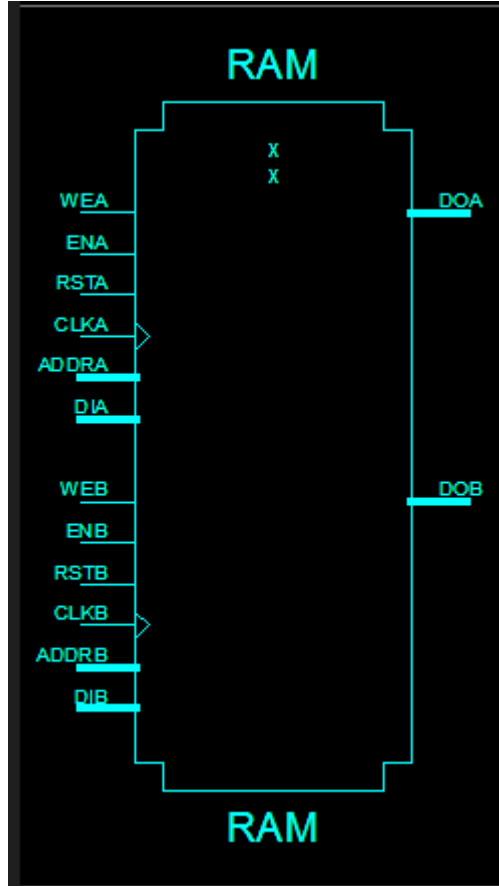


Figure 24 : RTL diagram of BRAM

Finally, the BRAM module successfully stored the image's pixel values from the hex file and showed its operation via simulation. The timing diagram depicted the BRAM's read and write operations, demonstrating the efficient storing and retrieval of data. The simulator's console display validated the precision of the BRAM module's functions. The RTL diagram also gave a visual image of the BRAM module's internal construction and connections. The successful installation and study of the BRAM module verifies its critical function in the co-processor architecture in storing image data and supporting future image processing operations.

4.2.2. Histogram Module

The histogram module is made up of RAM which has 4096 entries and contains 8 bits of data per entry. The address bus is 8 bits wide, allowing for a total of 256 distinct addresses. The RAM is similarly 8 bits wide, allowing for the storing of a single byte in each memory address.

Furthermore, the counter used to address the histogram module is 8 bits in size. This counter guarantees that each intensity value in the image is addressed and counted efficiently. The histogram module can properly determine the frequency distribution of pixel intensities in the image by incrementing the counter for each encountered intensity value.

The Histogram module, with its suitable size and addressing techniques, allows the co-processor to conduct efficient histogram computations and hence simplify future image improvement algorithms. The timing diagram for the Histogram module is shown in Figure 25. This graphic depicts the timing behavior of the module when it is in operation.

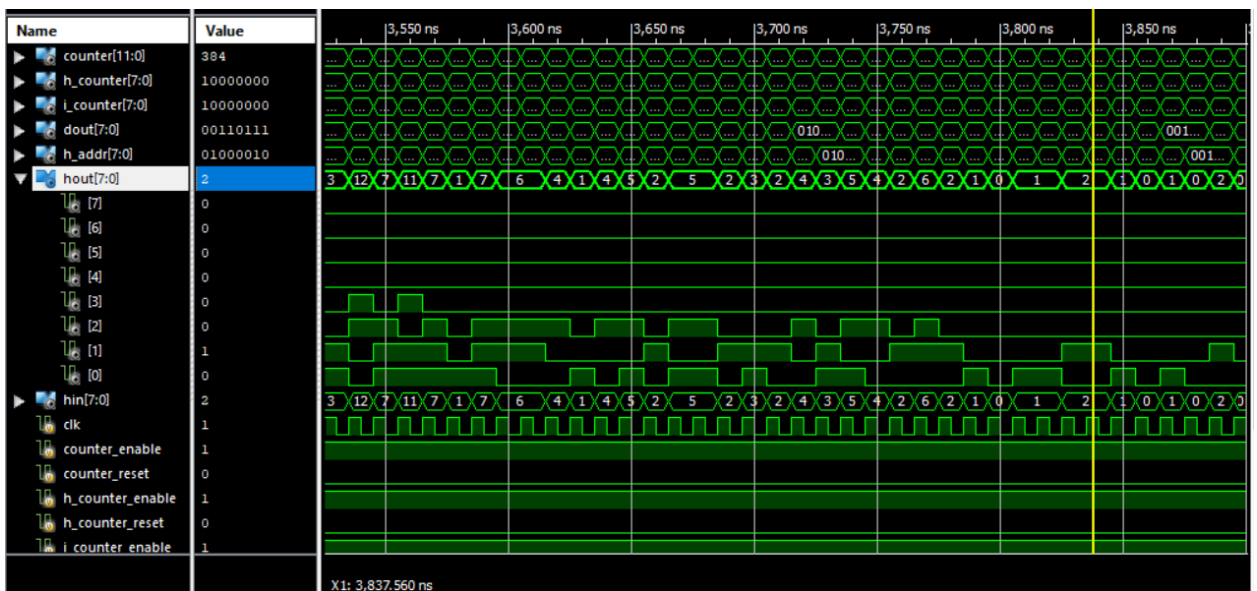


Figure 25 : Histogram Module Simulation

The RTL diagram of the histogram module is displayed in Figure below. The diagram provides a visual representation of the internal structure and connectivity of the histogram module.

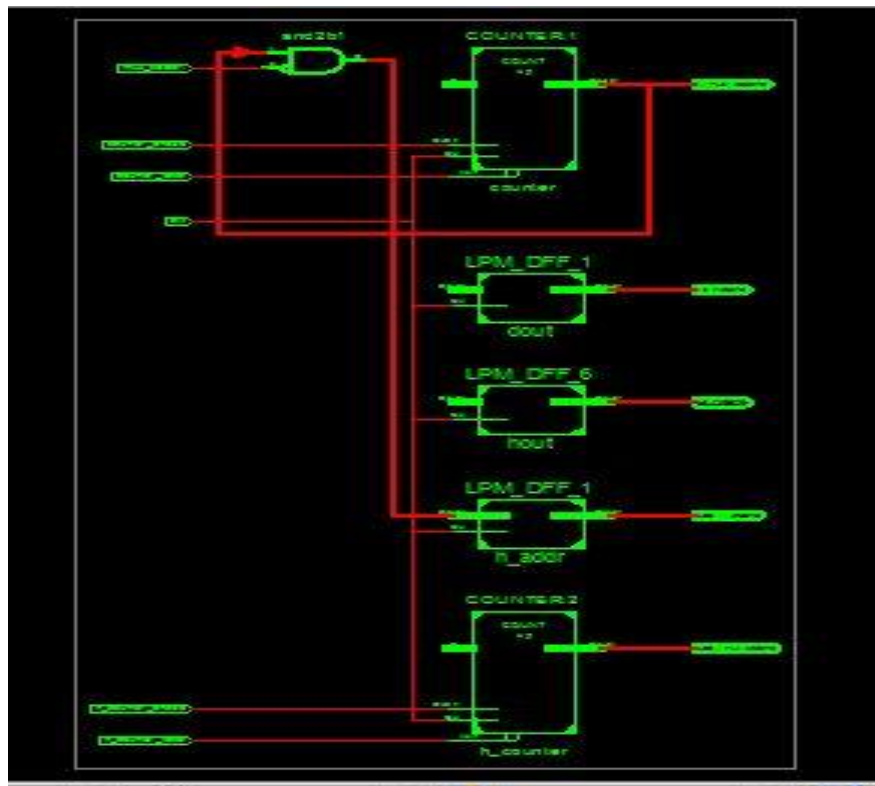


Figure 26 : RTL diagram

The console window output for the histogram module is displayed in Figure below. Analyzing the console window output helps in verifying the correct functionality and performance of the histogram module.

```

Console
-----
ISim M.53d (signature 0x7dea747)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim> |

```

Figure 27 : Simulation Console Result

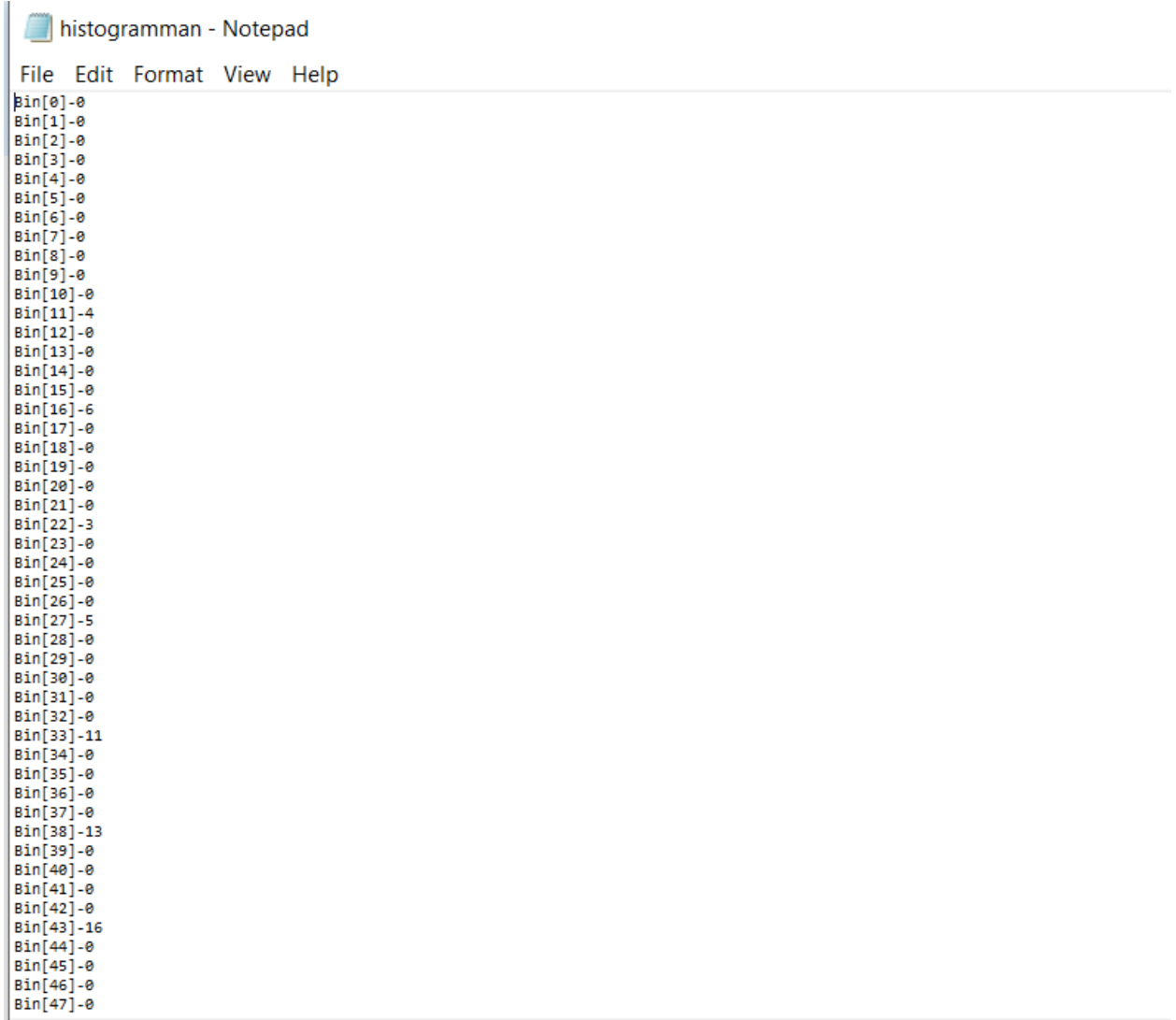
4.3. Software result Analysis: C Implementation

The software implementation of histogram calculation and histogram equalization in Vivado HLx using C resulted in successful image enhancement. The histogram technique calculated the frequency of each pixel intensity value in the supplied image precisely, giving useful information about the image's intensity distribution. This data was then used in the histogram equalization process to increase the image's contrast and visual quality.

4.3.1. Histogram Implementation

The C histogram was saved as a text file, enabling for simple representation and comparison of the histogram data. The frequency counts of each pixel intensity value were included in the text file, giving a succinct and organized representation of the histogram.

The accuracy and consistency of the C-generated histogram were evaluated by comparing it to the histogram produced via MATLAB. The MATLAB text file, which also included frequency counts of pixel intensity values, was used for comparison.



```
File Edit Format View Help
Bin[0]-0
Bin[1]-0
Bin[2]-0
Bin[3]-0
Bin[4]-0
Bin[5]-0
Bin[6]-0
Bin[7]-0
Bin[8]-0
Bin[9]-0
Bin[10]-0
Bin[11]-4
Bin[12]-0
Bin[13]-0
Bin[14]-0
Bin[15]-0
Bin[16]-6
Bin[17]-0
Bin[18]-0
Bin[19]-0
Bin[20]-0
Bin[21]-0
Bin[22]-3
Bin[23]-0
Bin[24]-0
Bin[25]-0
Bin[26]-0
Bin[27]-5
Bin[28]-0
Bin[29]-0
Bin[30]-0
Bin[31]-0
Bin[32]-0
Bin[33]-11
Bin[34]-0
Bin[35]-0
Bin[36]-0
Bin[37]-0
Bin[38]-13
Bin[39]-0
Bin[40]-0
Bin[41]-0
Bin[42]-0
Bin[43]-16
Bin[44]-0
Bin[45]-0
Bin[46]-0
Bin[47]-0
```

Figure 28 : Histogram generated by MATLAB

The similarity and deviation between the C-generated histogram and the MATLAB histogram were compared. The level of agreement between the two histograms was quantified using statistical measures such as mean squared error (MSE).

We tested the quality and accuracy of the C implementation by comparing the C-generated histogram to the MATLAB histogram.

histogramc - Notepad

File Edit Format View Help

```
Bin[0]-0
Bin[1]-0
Bin[2]-0
Bin[3]-0
Bin[4]-0
Bin[5]-0
Bin[6]-0
Bin[7]-0
Bin[8]-0
Bin[9]-0
Bin[10]-0
Bin[11]-4
Bin[12]-0
Bin[13]-0
Bin[14]-0
Bin[15]-0
Bin[16]-4
Bin[17]-0
Bin[18]-0
Bin[19]-0
Bin[20]-0
Bin[21]-0
Bin[22]-3
Bin[23]-0
Bin[24]-0
Bin[25]-0
Bin[26]-0
Bin[27]-5
Bin[28]-0
Bin[29]-0
Bin[30]-0
Bin[31]-0
Bin[32]-0
Bin[33]-11
Bin[34]-0
Bin[35]-0
Bin[36]-0
Bin[37]-0
Bin[38]-13
Bin[39]-0
Bin[40]-0
Bin[41]-0
Bin[42]-0
Bin[43]-16
Bin[44]-0
Bin[45]-0
Bin[46]-0
Bin[47]-0
```

Figure 29 : Histogram generated by C in vivado

4.3.2. Histogram Equalization

The histogram equalization technique implemented in C was tested to see how successful it was in enhancing the input image. The outcome analysis included comparing the original image to the histogram-equalized image in order to assess the benefits of the equalization technique.

The histogram equalization technique effectively redistributed the input image's pixel intensities, resulting in better contrast and visual improvement. This was proved by a visual comparison of the original image and the equalized image, which revealed a considerable increase in overall image quality.



Figure 30 : Original Image



Figure 31 : MATLAB output



Figure 32 : VIVADO output

Figure below presents the synthesis report, which provides detailed information and analysis of the histogram equalization module's implementation.

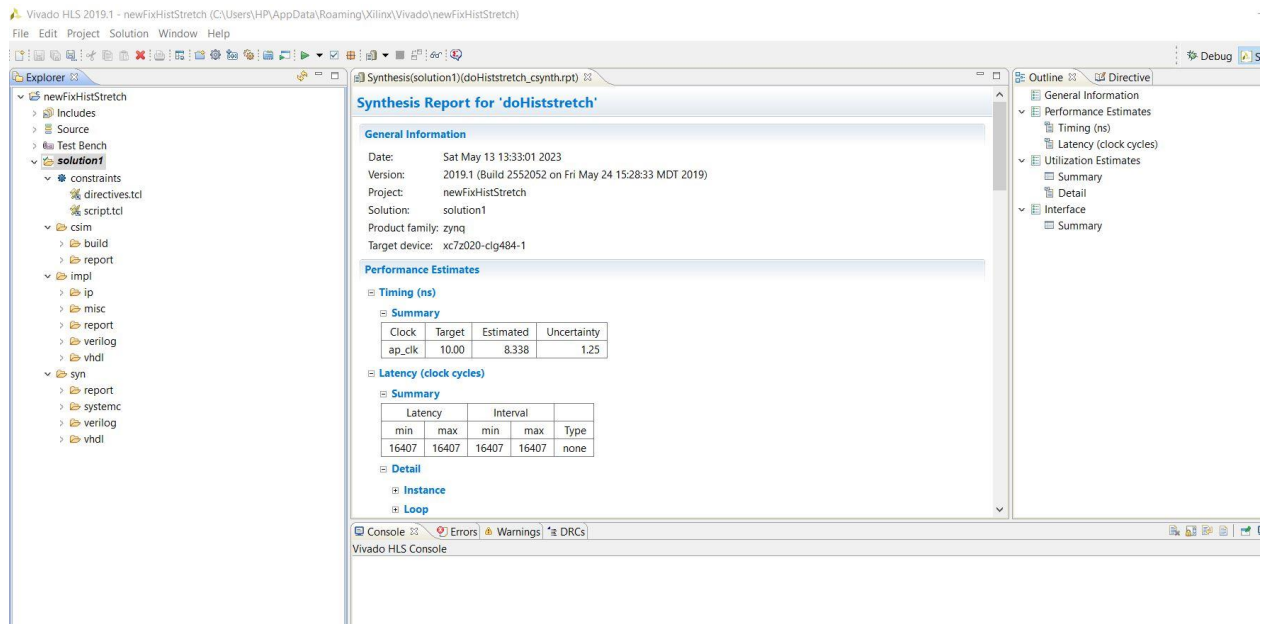


Figure 33 : Synthesis Report

The output of the console window during the execution of the histogram equalization module in the Vivado HLS environment is shown in the figure below, which shows the compilation process, simulation results, and any pertinent errors or cautions.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strate
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.0	0	0	5/18/23, 12:14 AM	00:00:47	Vivado Sy
impl_1	constrs_1	write_bitstream Complete!	0.971	0.000	0.020	0.000	0.000	1.443	0	7679	9767	4.0	0	1	5/18/23, 12:15 AM	00:03:49	Vivado Im
Out-of-Context Module Runs																	
design_3		Submodule Runs Complete													5/17/23, 11:49 PM	00:25:03	

Figure 34 : Simulation Console Window

4.3.3. Mean Square Error

As a quantitative measure of equalization quality, the Mean Square Error (MSE) was determined between the original image and the histogram-equalized image. A lower MSE value implies a smaller disparity between the original and equalized images, signifying better image detail preservation accuracy.

The MSE value analysis demonstrated that the C implementation of the histogram equalization technique produced a comparatively low MSE, suggesting high reconstruction quality. This implies that the equalized image was quite similar to the original image, with little information lost during the equalization process. The observed low MSE value demonstrates the histogram equalization algorithm's usefulness in properly dispersing pixel intensities and boosting image contrast.

4.4. IP core Integration

The histogram equalization module was included into the current hardware architecture as part of the IP core integration procedure. The goal of this integration was to easily connect the histogram equalization module's capabilities with other IP cores and system components.

By effectively integrating the histogram equalization IP core, we were able to create a coherent system in which the image processing capability functioned in tandem with other components. This integration guaranteed efficient data flow and allowed the image processing module to communicate with other system components.

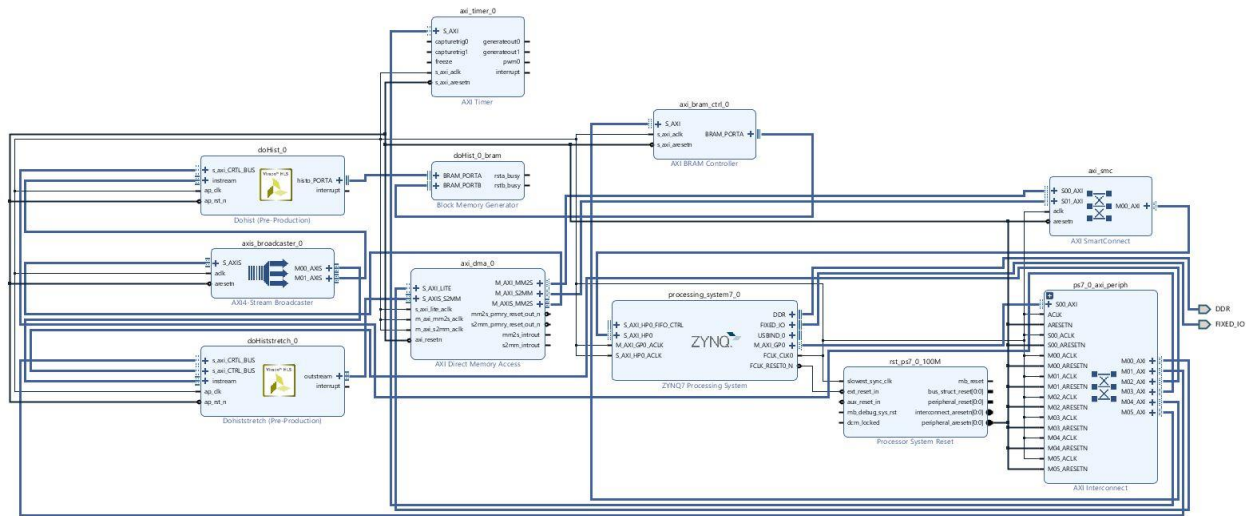


Figure 35 : IP core Integration

In Verilog, the IP core for the histogram block RAM is integrated by connecting it to the IP core of the histogram block RAM in C through port B. This link allows data to be sent seamlessly between the Verilog IP core and the C IP core, allowing for quick processing and storing of histogram data. AXI smart connect is also used to link port A of the histogram block RAM IP core to the ZYNQ processing system. This link permits data and control signals to be sent between the histogram module and the ZYNQ processing system.

The IP core "dohist" block RAM in C is also linked to the IP core "dohiststretch" module. This link allows data to flow from the histogram calculating IP core to the histogram stretching IP core, allowing the use of histogram equalization methods to increase image contrast and visual quality.

The system maintains smooth data and control signal flow by combining these IP cores and providing the appropriate connections, allowing for efficient execution of the histogram equalization procedure. This integration allows the Verilog-based histogram block RAM IP core to work in tandem with the C-based histogram IP core, as well as interaction between the histogram module and the ZYNQ processing system.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
doHistStretch_0	s_axi_CRTL_BUS	Reg	0x43C0_0000	64K	0x43C0_FFFF
doHist_0	s_axi_CRTL_BUS	Reg	0x43C1_0000	64K	0x43C1_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF
axi_timer_0	S_AXI	Reg	0x4280_0000	64K	0x4280_FFFF
axi_dma_0					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_...	0x0000_0000	512M	0x1FFF_FFFF
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_...	0x0000_0000	512M	0x1FFF_FFFF

Figure 36 : Address Editor

4.4.1. Bitstream generation

Following successful testing and validation of the integrated system, the production of the bitstream file is the next critical step. This procedure entails compiling the design and synthesizing it into a configuration file that contains the whole FPGA design.

The produced bitstream file is then put into the Arty Z7-10 board. The bitstream file is sent to the FPGA device during this programming stage, allowing it to execute the implemented design. The Arty Z7-10 board acts as the FPGA's target platform, providing the hardware resources required for functioning.

The FPGA is now ready to execute the implemented design and perform the expected duties after successfully creating and loading the bitstream onto the Arty Z7-10 board. The bitstream file connects the synthesized design to the real hardware, enabling for the smooth integration and execution of complicated algorithms and features.

4.5. Hardware Interfacing

The hardware interface step was a success, allowing the design and the Arty Z7-10 board to communicate and integrate seamlessly. A programming cable was used to connect the board to a PC, enabling a dependable interface for data transmission and setup. This connection

permitted the transmission of the bitstream file from the PC to the board and allowed for fast programming of the FPGA.

CHAPTER # 05

FUTURE WORK

5.1. REAL TIME IMAGE ENHANCEMENT

5.2. OPTIMIZATION FOR PARALLEL PROCESSING

5.3. INTEGRATION WITH MACHINE LEARNING TECHNIQUES

5.4. ADVANCED IMAGE ENHANCEMENT ALGORITHM

5.5. ENERGY EFFICENCY AND POWER OPTIMIZATION

5.6. SYSTEM INTEGRATION AND DEPLOYMENT

Future Work

The future work for the project of creating a co-processor for image enhancement included investigating several options for development and extension. This might involve improving the co-capabilities processors and performance via numerous routes of research and development. The following are some potential areas for additional co-processor enhancement:

5.1. Real-Time Image Enhancement:

Future work might include implementing real-time image improvement via HDMI input and output. This would include combining the planned co-processor with HDMI ports to allow for direct processing and presentation of live video feeds. Real-time image augmentation would improve the system's usability and practicality by providing instant visual input and use in a variety of real-time imaging applications.

5.2. Optimization for Parallel Processing:

Future work might concentrate on refining the architecture for parallel processing to improve the co-performance processors even more. This might include investigating parallel architectures, such as using several processing units or adopting parallel processing methods, in order to accelerate image enhancing algorithms and increase total processing speed.

5.3. Integration with Machine Learning Techniques:

Another direction for future research is to include machine learning methods into the image enhancing process. The co-processor may learn and adapt to varied image features by implementing machine learning models or algorithms, resulting in more sophisticated and tailored enhancing outcomes. This would include training the system on massive datasets and implementing effective real-time inference techniques.

5.4. Advanced Image Enhancement Algorithms:

The project may be developed further to investigate and implement more complex image enhancing methods. Techniques such as multi-scale processing, local contrast enhancement, noise reduction, and adaptive enhancement depending on image content might be included. Implementing and enhancing these algorithms on the co-processor

would offer a greater variety of image augmentation capabilities while also improving image quality.

5.5. Energy Efficiency and Power Optimization:

Future development might concentrate on enhancing the co-energy processor's efficiency and power optimization. This might include investigating low-power design methodologies, adopting power management systems, and optimizing the hardware architecture to reduce power consumption while maintaining performance. Improving the coprocessor's energy efficiency would help sustainable computing and lessen the environmental effect of image processing operations.

5.6. System Integration and Deployment:

Once the co-processor architecture has been refined, further work would include integrating it into a whole system and doing extensive testing and validation. This would include creating a user-friendly interface, assuring interoperability with various platforms and software frameworks, and assessing the system's performance in real-world circumstances. To guarantee the effective installation and acceptance of the image enhancement system, deployment factors such as scalability, reliability, and user feedback should be addressed.

CHAPTER # 06

CONCLUSION

6.1. OVERVIEW

6.2. OBJECTIVES ACHIEVED / ACHIEVEMENTS

6.3. CONTRIBUTIONS

6.4. LIMITATIONS

6.5. APPLICATIONS

6.5.1. MEDICAL IMAGING

6.5.2. COMPUTER VISION

6.5.3. REMOTE SENSING

6.5.4. VIDEO PROCESSING

6.5.5. EMBEDDED SYSTEMS

Conclusion

6.1. Overview:

The goal of this project was to build histogram equalization on an FPGA using Verilog and C and then integrate it with the Arty Z7-10 board. The method was developed and tested in MATLAB for the project. The design includes BRAM and histogram calculating units. The C software implementation enabled result comparison. The integration step used block RAM IPs and AXI smart connect to link the FPGA design to the Arty Z7-10 board. Successful hardware interfacing and bitstream loading demonstrated project completion.

6.2. Objective Achieved/ Achievements:

This project's accomplishments include improving resource use in Verilog and C designs inside Xilinx and Vivado, which resulted in efficient implementation. The FPGA design included modules for BRAM and histogram computation, allowing for efficient storage and processing. The use of software permitted result comparison and confirmation, assuring accuracy and dependability. Overall, this project was effective in using resources efficiently and integrating FPGA modules for histogram equalization.

6.3. Contributions:

The project demonstrates how to leverage FPGA technology for efficient image processing tasks like histogram equalization. It contributes to the development of sophisticated image processing algorithms by proving the usefulness of FPGA-based solutions. It emphasizes the significance of combining hardware and software components for best performance. The employment of IP cores with software implementation allows for a synergistic approach, resulting in greater efficiency and accuracy in image processing applications. It stresses the effective use of hardware resources in FPGA designs, such as block RAM. It adds to resource efficiency and allows quicker and more efficient image processing methods by building modules that

efficiently store and analyze image data.

6.4. Limitations:

FPGAs' resources, such as block RAM and logic components, are restricted. The size of the input images or the number of actions that may be done concurrently may be limited depending on the complexity of the implemented modules and the available resources. As a result, the project's scalability and applicability to larger-scale image processing applications may be limited.

6.5. Applications:

Some of the key areas where this project can be applied include:

6.5.1. Medical Imaging:

Histogram equalization may be used to improve medical imaging, assisting in the identification and analysis of numerous medical disorders. It may increase image contrast and emphasize key characteristics, allowing healthcare providers to make more accurate diagnoses and treatment plans.

6.5.2 Computer Vision:

Image enhancing methods like histogram equalization are essential in computer vision applications. They may be used to improve image quality, eliminate noise, and extract more features. The FPGA-based implementation of this project allows for real-time processing, making it suited for computer vision applications such as object detection, tracking, and surveillance systems.

6.5.3. Remote Sensing:

Image enhancement methods are often used in remote sensing applications such as satellite imaging and aerial photography to improve viewing and analysis of the Earth's surface. FPGA-based systems benefit from fast processing, allowing for real-time improvement of large-scale remote sensing photos.

6.5.4. Video Processing:

Image processing systems based on FPGAs may also be used for video processing applications such as real-time video enhancement, object tracking, and video

surveillance. FPGA-based BRAM and histogram modules' efficient storage and processing capabilities allow for smooth incorporation into video processing pipelines.

6.5.5. Embedded Systems:

Image processing based on FPGAs may be incorporated into embedded systems for applications like robotics, autonomous cars, and industrial automation. FPGAs are ideal for resource-constrained embedded situations because of their real-time processing capabilities, small size, and low power consumption.

This project contributes to the improvement of real-time image processing applications in numerous domains by utilizing the FPGA-based implementation of histogram equalization and associated modules, boosting image quality, allowing correct analysis, and simplifying decision-making processes.

BIBLIOGRAPHY

i. REFERENCES

REFERENCES

- [1] Johnston, Phillip. “Simple Fixed-Point Conversion in C.” *Embedded Artistry*, 12 July 2018, embeddedartistry.com/blog/2018/07/12/simple-fixed-point-conversion-in-c/. Accessed 19 May 2023.
- [2] Hall TS, Hamblen JO (2004) System-on-a-programmable-chip development platforms in the classroom. *IEEE Transactions on Education*: 47(4): 502–507
- [3] WISHBONE System-on-Chip (SoC) interconnection architecture for portable IP cores. (2002) B.3. <http://www.opencores.org/projects.cgi/web/wishbone/wishbone>. Accessed 21 May 2008.
- [4] “Arty Z7: Zynq-7000 SoC Development Board.” *Digilent*, digilent.com/shop/arty-z7-zynq-7000-soc-development-board/. Accessed 19 May 2023.
- [5] Tardi, Carla. “ASIC Bitcoin Miner Definition.” *Investopedia*, 16 Mar. 2022, www.investopedia.com/terms/a/asic.asp.
- [6] “SoCs, MPSoCs and RFSocCs.” *Xilinx*, www.xilinx.com/products/silicon-devices/soc.html.
- [7] Ltd, Arm. “Microprocessor Cores and Processor Technology – Arm®.” *Arm | the Architecture for the Digital World*, www.arm.com/products/silicon-ip-cpu.
- [8] panupriyaa7. *Point Processing*. 25 Oct. 2016, www.slideshare.net/panupriyaa7/point-processing-amp-gray-level-transformations.
- [9] “Doulos.” *Www.doulos.com*, www.doulos.com/knowhow/verilog/rtl-verilog/#:~:text=RTL%20is%20an%20acronym%20for. Accessed 19 May 2023.
- [10] “Neighborhood and Block Processing - MATLAB & Simulink.” *Www.mathworks.com*, www.mathworks.com/help/images/neighborhood-and-block-processing.html. Accessed 19 May 2023.
- [11] “4.4. Global Image Processing Concepts.” *Mevislabdownloads.mevis.de*, mevislabdownloads.mevis.de/docs/current/MeVisLab/Resources/Documentation/Publish/SDK/MLGuide/ch04s04.html. Accessed 19 May 2023.

- [12] R. C. Gonzalez and R. E. Woods, Digital Image Processing, Third Edition, 2008.
- [13] “Histograms - Understanding the Properties of Histograms, What They Show, and When and How to Use Them | Laerd Statistics.” *Statistics.laerd.com*, statistics.laerd.com/statistical-guides/understanding-histograms.php#:~:text=A%20histogram%20is%20a%20plot.
- [14] “Cumulative Distribution Function - an Overview | ScienceDirect Topics.” *Www.sciencedirect.com*, www.sciencedirect.com/topics/mathematics/cumulative-distribution-function.
- [15] “Cumulative Distribution Function.” *Probabilitycourse.com*, 2020, www.probabilitycourse.com/chapter3/3_2_1_cdf.php.
- [16] “Probability Density Function | PDF | Distributions.” *Www.probabilitycourse.com*, www.probabilitycourse.com/chapter4/4_1_1_pdf.php.
- [17] Frost, Jim. “Mean Squared Error (MSE).” *Statistics by Jim*, 12 Nov. 2021, statisticsbyjim.com/regression/mean-squared-error-mse/.
- [18] “Documentation – Arm Developer.” *Developer.arm.com*, developer.arm.com/documentation/ka003292/latest#:~:text=The%20Intel%20HEX%20file%20is. Accessed 19 May 2023.
- [19] “BRAM and Other Memories.” *Www.xilinx.com*, www.xilinx.com/htmldocs/xilinx2017_4/sdaccel_doc/jbt1504034294480.html.
- [20] “Multiplexer (MUX) and Multiplexing Tutorial.” *Basic Electronics Tutorials*, Aug. 2013, www.electronics-tutorials.ws/combinational/comb_2.html.
- [21] Moore, Karleigh, and Dishant Gupta. “Finite State Machines | Brilliant Math & Science Wiki.” *Brilliant.org*, 2011, brilliant.org/wiki/finite-state-machines/.