DE-42 (EE) Abdullah Benyamin, Muhammad Kashif, Nauman Saeed, Zoraiz Ahmad

FPGA Implementation of Optimized Deep Learning Algorithm

COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY RAWALPINDI
2024

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-42 EE
PROJECT REPORT

FPGA Implementation of Optimized Deep Learning Algorithm

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

Bachelor of Engineering
in
Electrical
2024

Submitted By:

Abdullah Benyamin
Muhammad Kashif
Nauman Saeed
Zoraiz Ahmad

CERTIFICATE OF APPROVAL

It is to certify that the project **"FPGA Implementation of Optimized Deep Learning Algorithm"** was done by **NS Abdullah Benyamin**, **NS Muhammad Kashif, NS Nauman Saeed** and **NS Zoraiz Ahmad** under the 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. Abdullah Benyamin

   NUST ID: 346935 Signature: _____

2. Muhammad Kashif

   NUST ID: 332352 Signature: _____

3. Nauman Saeed

   NUST ID: 346173 Signature: _____

4. Zoraiz Ahmad

   NUST ID: 348656 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 theseverity of the offense.

1. **Abdullah Benyamin** _____

2. **Muhammad Kashif** _____

3. **Nauman Saeed** _____

4. **Zoraiz Ahmad** _____

COPYRIGHT STATEMENT

ACKNOWLEDGMENTS

We like to express our heartiest gratitude and appreciate all the individuals and organizations who have contributed to the successful completion of this project. Their guidance, encouragement and support were invaluable throughout this journey.

First and foremost, we would like to thank our project supervisor **Dr. Usman Ali** and Co-Supervisor **Asst**. **Prof. Kamran Aziz Bhatti,** whose guidance, expertise and continuous encouragement played crucial role in development of this project. Their insightful suggestions and constructive feedback have greatly contributed to overall quality of this project.

We extend our sincere thanks to the faculty members of the College of Electrical and Mechanical Engineering, NUST, especially to the Department of Electrical Engineering, for providing us an excellent academic research environment. Their dedication to imparting knowledge and fostering innovation has been instrumental in our learning and growth.

We are grateful to our fellow classmates especially Saifullah Masud, Muhammad Umar Farooq, Sajjad Ali and Masoom Raza and colleagues for their assistance, collaboration, and brainstorming sessions throughout this project. Their collective efforts and diverse perspective have greatly enhanced our understanding and problem-solving skills.

We would also like to thank the technical staff at College of Electrical and Mechanical Engineering, NUST for the support and guidance received from them. Their expertise in hardware implementation and software development has been instrumental in the smooth execution of our project. Also, for helping us in utilizing their laboratory facilities.

We express our gratitude to the participants of our study who generously volunteered for their time and cooperation in this project. Their involvement has been crucial in collecting the necessary data and validating the effectiveness of our project.

Lastly, we would like to express our gratitude to our relatives and allies for the consistent aid they have been providing us with all through this course of action. It is their fondness, motivation and trust in our own competencies that has given us the zeal to conquer any obstacles along the way.

Even if it is not feasibly realistic to name each person individually, we are grateful for the cumulative input of all those who have contributed to this project. It would not have been achieved without their continuous help and dedication.

Thank you all for being an integral part of our journey and for your invaluable contributions to the success of this project.

Abdullah Binyamin

Muhammad Kashif

Nauman Saeed

Zoraiz Ahmad

ABSTRACT

Deep learning has significantly improved image recognition, especially for digit classification tasks. However, the substantial computational requirements of deep learning models limit their use in real-time scenarios. This project addresses these challenges by leveraging the unique strengths of Field Programmable Gate Arrays (FPGAs), which are highly parallelizable and customizable, making them suitable for high-performance computing applications. In particular, this research explores implementing a Convolutional Neural Network (CNN) on an FPGA to accelerate digit recognition and evaluates the trade-offs between computational efficiency and model accuracy.

The main objective was to create and optimize a CNN capable of efficiently recognizing digits 0 through 9, leveraging the FPGA's ability to perform fast calculations with minimal power consumption. Advanced optimization techniques were employed to maximize resource utilization and improve data transfer efficiency, adapting the CNN architecture to fit within the FPGA's resource limits. This involved quantizing the model to fixed-point format, reducing the complexity of the computations while maintaining high accuracy.

The CNN was initially trained using Python, achieving a 95% accuracy rate after 30 epochs with a batch size of 86. The model was then converted to MATLAB for fixed-point quantization before being implemented in Verilog for deployment on the FPGA. This multi-step process ensured that the model retained its performance characteristics while being optimized for FPGA hardware.

Testing the FPGA-based CNN with the same dataset used during training demonstrated a high accuracy rate of 94%, closely matching the Python implementation. Resource utilization on the FPGA was carefully monitored, ensuring efficient use of Look-Up Tables (LUTs), Flip-Flops (FFs), and Digital Signal Processing (DSP) blocks. This highlights the FPGA's suitability for efficient and accurate digit recognition applications.

SUSTAINABLE DEVELOPMENT GOALS

SDG 4: Quality Education:

FPGA technology is used by our project to employ convolutional neural network which changes images from into words, harnessing deep learning in order to improve education accessibility. Converting visual information to written texts facilitates learning communities' inclusiveness which may allow for obtaining of information in multiple formats. In line with the Sustainable Development Goal 4, this method supports inclusive and equitable quality education and promotion of lifelong learning opportunities. Our aim is to enhance learning experiences through incorporation of high level technological aspects in educational setups as well as ensuring equal availability of information thereby achieving the broader objectives of SDG 4.

SDG 9: Industry Innovation and Infrastructure:

Our project uses FPGA tech to revolutionize educational technology, and finally convert visual data into text through the use of a convolutional neural network. It is in line with Sustainable Development Goal 9, which encourages infrastructure development and fostering sustainable industrialization. Technological infrastructure can be improved by making sure that FPGA is integrated into instructional equipment used in educational institutions thus supporting an innovative ecosystem for sustainable industrial growth. This allows developing more dynamic industries responsive to future learning needs.

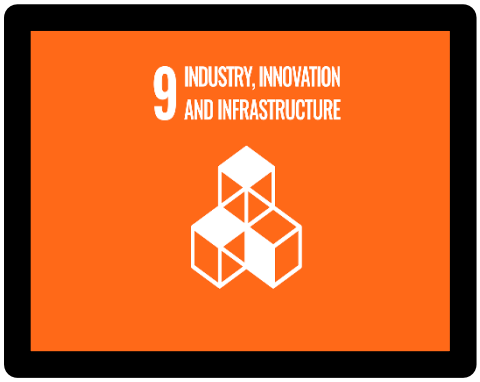

TABLE OF CONTENTS

Table of Contents

LIST OF FIGURES

Chapter 1:

Introduction

Problem Statement

The problem with using a Field Programmable Gate Array (FPGA) to execute Convolutional Neural Networks (CNNs) for digital image recognition is that they have complicated hardware constraints. On the other hand, even if CNNs are effective in tasks like handwriting recognition, there are still issues to deal with when it comes to FPGAs such as limited resources regarding processing and memory. These limits can greatly affect the power of computation and accuracy of a CNN system thereby making it difficult to meet real-time requirements including automatic data entry or assistive technologies. The goal of this final year project is to design and optimize an FPGA-based CNN that would recognize static **images** containing digits from 0-9.. The main difficulty will involve adjusting the CNN architecture so as to achieve maximum computational and power efficiencies on the FPGA without sacrificing its accuracy. It involves engineering solutions which can precisely balance between the computational requirements of a CNN and what an FPGA's hardware can actually handle. Consequently, this project intends to prove that FPGAs may perform many complicated deep learning operations effectively, therefore their use in real-time processing situations may become possible, thereby improving their adaptability.

Objectives:

To optimize the architecture of a CNN for efficient operation within FPGA resources is the primary goal. The following are part of this; minimizing power consumption, maximizing processing speed as well as maintaining high accuracy that is suited to the specific capabilities of FPGA technology. An effective and reliable solution for processing digit images directly on an FPGA with quick response times must be developed and integrated as it would be essential in realizing goals of this project.

Besides this, the system will be thoroughly tested to examine its performance in terms of accuracy, processing speed and power consumption. This test will compare FPGA based approach to conventional techniques for performing computations so that we can highlight where it is better and how it can be improved further. Advanced quantization techniques coupled with optimization methods will be investigated into and exploited to minimize computational complexity in CNN while maintaining its efficiency without noticeable loss of precision.

The project aims at showcasing FPGA's capability in handling complex deep learning tasks. These will be displayed through an optimized image recognizer on the system, stressing the merits of FPGAs over typical hardware platforms such as CPUs and GPUs in terms of fast processing speed and energy used up. Lastly, extensive documentation of how it was developed will be prepared together with user guidelines for future researchers or developers who would want to duplicate or modify the system for other uses. This document is therefore a useful material for extending the application of FPGA in deep learning beyond this project.

Significance of the Study:

The importance of this study is its ability to aid in the adoption of deep learning on FPGAs specifically for optimizing and deploying Convolutional Neural Networks (CNNs) in digit recognition. The uniqueness of these devices lies in their parallelism and programmability and this project has been able to show how they can assist to develop better and faster image recognition systems. When implemented successfully on an FPGA, CNN shows that FPGAs can be used for complex computational tasks, which have implications not only on the practicality of using FPGAs for complex computational tasks but also its relevance towards hardware accelerated learning, in real world applications. Industries such as automation data entry, security and aids need rapid accurate image processing. In addition, it provides a pointing out for further studies in the area that may result into more energy-efficient and cost-effective solutions within AI and ML domains. In going beyond what has been already achieved with FPGA-based architecture under deep learning, this study takes a valuable place among other works also concerned with technology optimization towards better performance or sustainability.

This digit recognition is only one example however the scope of this project is far beyond this example and it can be used for many other applications. It can be trained to classify 10 different images they can be of any object or different faces.

Further this project can be extended for any type of image recognition just by changing the values of weights in Surveillance Drone images and CCTV camera and Video feed from Electric Vehicles.

Chapter 2:

BACKGROUND AND LITERATURE REVIEW

Introduction:

A crucial leap in computational technology has been made with the integration of deep learning algorithms into hardware platforms specifically Field Programmable Gate Arrays (FPGAs). For instance, they are reconfigurable, function at very high speeds and consume less energy as compared to other options. This makes FPGAs ideal for edge computing workloads that need low latency and power consumption. Recent studies have focused on optimizing Convolutional Neural Networks (CNNs) which are used extensively in image and video processing applications for better utilization of FPGA characteristics. Among such improvements is quantization and simplification of convolutional operations to suit the limited computational resources available on FPGAs while still maintaining accuracy. According to a literature review despite the achievements made so far in deploying deep learning models on FPGAs, there is still a range of obstacles that leave this field open for more research including: resource control, real-time performance boosting and retaining accuracy with recent models. As such, this paper seeks to develop an optimized CNN implemented specifically on an FPGA platform for real time digit recognition.

Convolutional Neural Networks (CNNs) are commonly employed in video and image processing tasks making them good fit for FPGAs. The adaptability inherent in FPGA can be used to change CNN architectures so as to better match the characteristics of the underlying hardware, thus helping enhance computational efficiency and operational speed. Recent advancements have been geared towards model compression, advanced quantization techniques and simplification of convolution operations. The above optimizations aim at addressing limitations associated with less available computational resources in FPGAs compared to traditional CPUs or GPUs, while trying to keep or even surpass model accuracy.

Deep Learning on FPGA:

Integrating deep learning algorithms and Field Programmable Gate Arrays (FPGAs) is a major step in the area of edge computing and real-time processing technologies. This is as FPGAs are uniquely positioned to do so because they have high flexibility, configurability and efficient parallel processing. These features enable these chips to perform high-speed computations that are required by deep learning applications, but with less power than traditional CPUs or GPUs. Complex matrix operations and numerous layers of neurons make deep learning models such as Convolutional Neural Networks (CNNs) or to be computationally intensive. Still it is possible to let FPGAs easily accomplish this task thereby providing a better adaptation for hardware specificities of particular neural networks. Thus there are systems-on-chip that can simultaneously run several processes allowing fast calculations.

The use of FPGAs has gained traction in the management of computationally demanding models deployed in deep learning, examples being Convolutional Neural Networks (CNNs) and Recurrent Neural Networks. Their design is characterized by a good number of layers that work together for the intended goal. Nonetheless, traditional processors may find it quite difficult to handle such models

because they have many complex matrix operations. Nonetheless, FPGAs can be designed so as to facilitate these forms of computation. If an FPGA executes various operations at once, then it was programmed to perform convolution as well as many other tasks using parallelism.

Conversely, its programmability and ability to be programmed based on different algorithms or tasks suggest that FPGAs are increasingly being recognized for their potentiality in deep learning. Thus one FPGA could run numerous processing steps and inference stages according to what changes are occurring in algorithms or priorities assigned to where different works must be done. For instance, real-time video processing, mobile computing, autonomous driving and any other application that requires dynamic computing.

Benefits of Implementing Deep Learning on FPGA:

One advantage of using FPGAs for deep learning is that they are programmable, meaning developers can upgrade hardware algorithms without any physical changes. This adaptability is crucial when deploying systems out in the field where new data may call for an update of the algorithms or merely refining their performance while at the same time tweaking them. If say deep learning models are constantly changing, they may need different computational resources or be better improved through newer algorithms that consume less power or enhance processing speed. For this reason, implementation on FPGA allows for adaptation to these hardware and thus extends the lifetime of a device which in turn reduces costs associated with hardware refresh cycles.

In addition, reprogramming FPGAs can greatly reduce the development cycle of deep learning applications. A prototype can be developed by programmers using FPGA, and then test it under actual conditions before its update based on performance numbers and user feedback instead of modification through hardware iterations. In this way, not just does this approach save money but it also speeds up the introduction of new attributes. Some types of deployment, such as self-driving cars, adaptive signal processing and real-time surveillance systems, involve changing conditions or data characteristics over time and are priceless when their processing algorithms can be remotely updated on FPGAs.

Challenges in Implementing Deep Learning on FPGA:

In view of these advantages, there are a number of challenges associated with deploying deep learning on FPGAs. They include the FPGA development and programming complexities that usually require the knowledge of hardware description languages like VHDL or Verilog. In addition to this, FPGAs might have an inadequate amount of on-chip memory which may act as a bottleneck in cases where there is need to process larger neural networks at once requiring much more data.

The complexity involved in programming Field Programmable Gate Arrays (FPGAs) is another deterrent that one faces. While other programming environments tend to focus straightly on CPUs or GPUs, developers working with FPGA must have some background understanding about hardware description languages like VHDL or Verilog which normally have steep learning curve and differ greatly from software programming paradigms.

Moreover, for FPGAs to be optimized such that they can run deep learning models, it entails converting high-level abstractions/operations into low-level hardware instructions.

Hence, deep learning is very rapidly rendering FPGA implementations obsolete. For instance, the

introduction of new neural network architectures and training techniques may not be compatible with existing FPGAs.

Hardware Acceleration:

To avoid these complications, hardware acceleration methods are commonly employed. This involves making certain areas inside the FPGA specialized to increase specific functions' efficiency over those achievable by general-purpose processors. For example, in such a case, customized IP cores can be made to multiply or do convolution faster by designing them explicitly for those parts of computations in a neural network.

Therefore, FPGAs are able to accommodate such algorithms comprehensively considering issues of flexibility, performance as well as power efficiency. They are best suited for real-time applications including autonomous vehicles, medical imaging systems and intelligent surveillance systems among others. Subsequently, several new approaches have emerged on optimizing deep learning implementations on FPGA's.

There is an importance of hardware acceleration techniques in enhancing deep learning on FPGAs for better performance on some operations such as multiplication and convolution which are key in neural network computations.

This is further possible with FPGA-specific libraries and tools that simplify the design process so that software programmers without extensive hardware design experience can be able to do it easily. The case in point is the fact that using high-level synthesis (HLS) tools, developers can write algorithms using widely popular programming languages like C or C++ and which are then automatically translated into hardware description languages thus reducing the gap between software and hardware design.

In addition, recent develop in reconfigurable field programmable gate arrays have provided them with flexibility in terms of variation. They are now able to shift their hardware configurations so that only parts of the circuitry are involved in specific tasks while the other parts continue working. This implies that on-the-fly changes of hardware configurations due to real-time requirements of the neural network as it processes data result into more effective utilization of resources. Energy consumption can be minimized as well as performance enhanced by a dynamic reconfiguration through which FPGA can optimize itself for any given task.

Sr.No	Title of Paper	Author
1	Neural Network Implementation Using FPGA Issues and Application	A. Muthuramalingam S. Himavathi
2	On the Implementation of Fixed-Point Exponential Function for Machine Learning and Signal Processing Accelerators	Mahesh Chandra
3	SIGN LANGUAGE RECOGNITION USING 3D CONVOLUTIONAL NEURAL NETWORKS	Jie Huang Houqiang Li Wengang Zhou Weiping Li

Chapter 3:

METHODOLOGY

Introduction:

This chapter highlights the methods used in implementing and optimizing a convolutional neural network (CNN) for digit recognition on an FPGA. These include: Creating an initial data set, writing appropriate software using high level programming languages which is then converted into hardware description language (HDL) for the purpose of FPGA implementation.

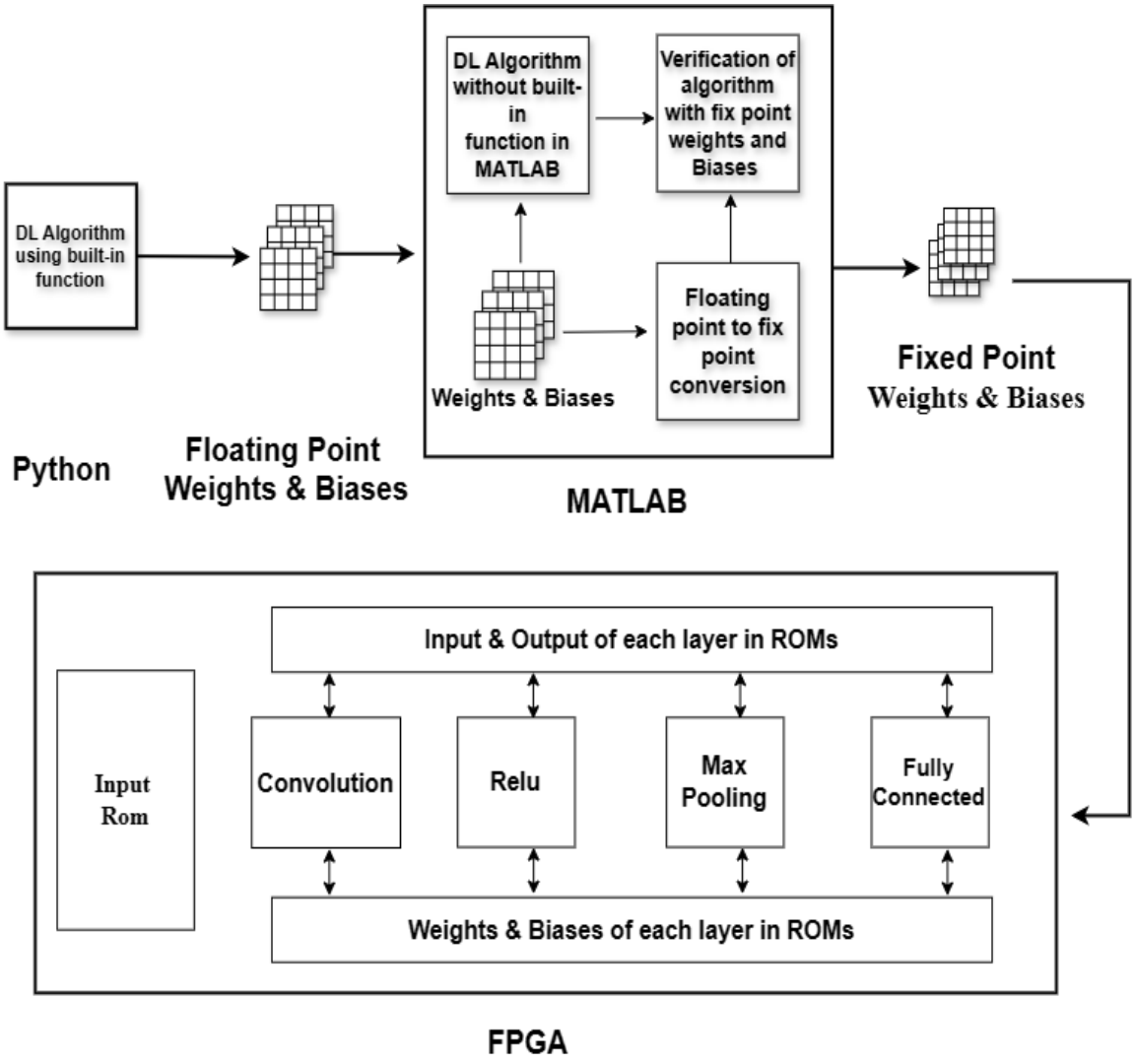

Figure 1: Block Diagram of Complete Project

Data Preparation:

The dataset of MNIST used in project, is a large collection of handwritten numbers and it also has been a standard benchmark in image recognition. Composed with roughly 25,000 to 30,000 samples of numbers from zero through nine on various handwriting, this latest version of the MNIST data set is preferred because it offers comprehensive coverage. The dataset was chosen because it's extensive and can be used by machine learning algorithms to identify different kinds of handwriting. This new version has about 25,000 – 30,000 instances of written numerals fairly distributed among all the ten groups. The images are rescaled to be small (28 × 28 pixels) this makes further automated recognition easier since preprocessing steps that must be done before training are reduced. Every picture in the MNIST database is actually an ordinary black-and-white bitmap where a grey scale level between white (0) and black (255) is assigned to each pixel. Such diversity allows deep models for recognizing smaller variations in handwriting styles.
In other words, the MNIST dataset's pervasiveness within the machine learning community has established it as a conventional barometer when it comes to judging how well different image recognition algorithms perform. Through frequent application in diverse research and developmental undertakings, it is possible to evaluate relative performance of models and architectures, which can be seen as a stepping stone towards progressive developments in digital number identification technology. This previous background makes the MNIST dataset very useful for training and testing our CNN model optimization on FPGA in order to achieve high accuracy numbers recognition.

Algorithm Development in Python:

The CNN architecture is equipped with various important features that are specific to optimizing the task of digit recognition, built on the original MNIST dataset. These layers within CNN have been tailored to extract and understand input digit images in a better way.

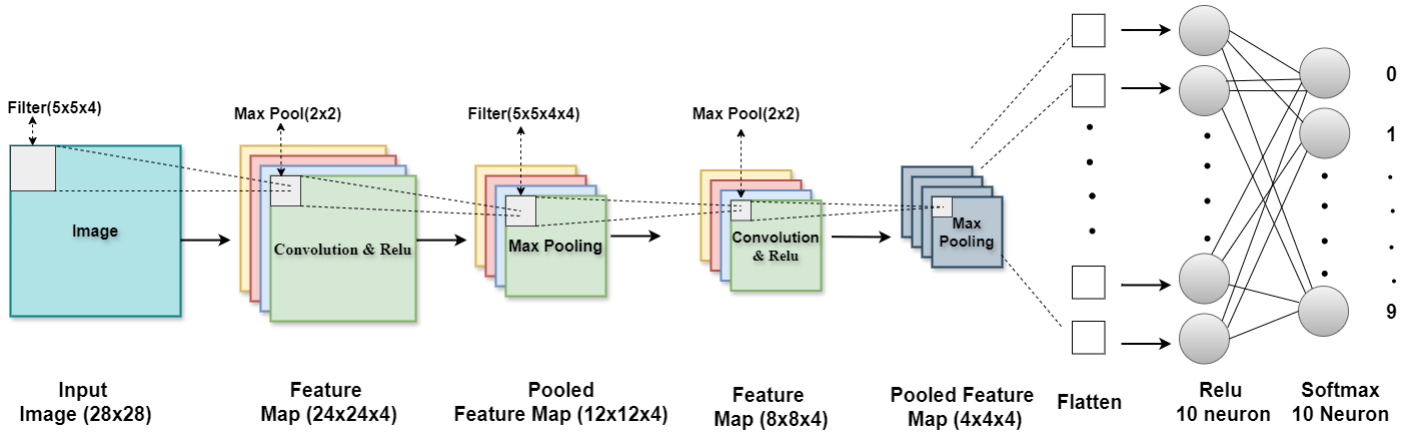

Figure 2: Visual Representation of CNN Model

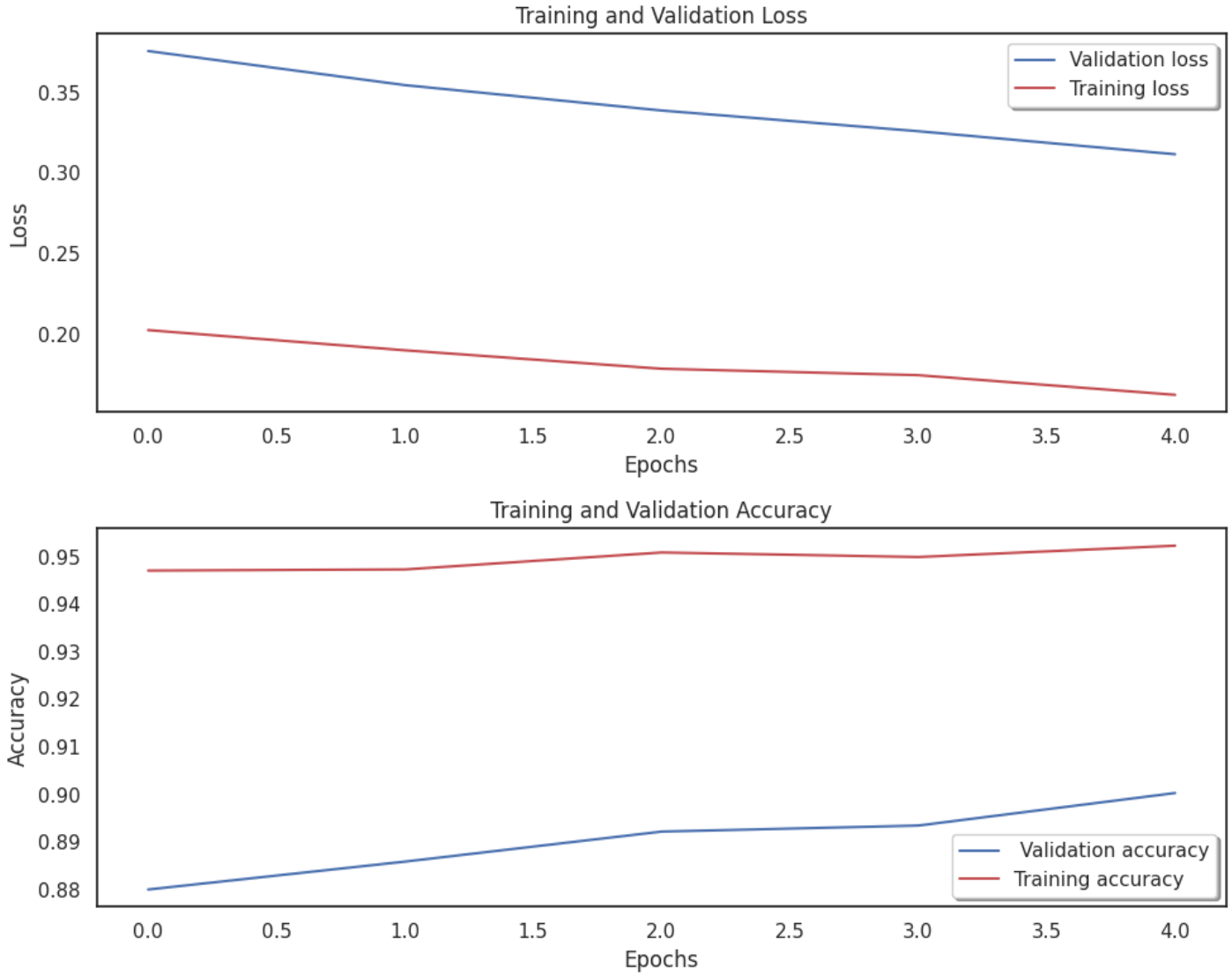

Figure 3: CNN Model Training and Validation

Translation to MATLAB:

The transition from Python to MATLAB played a pivotal role in refining the convolutional neural network (CNN) for FPGA implementation. This translation was instrumental for multiple reasons:

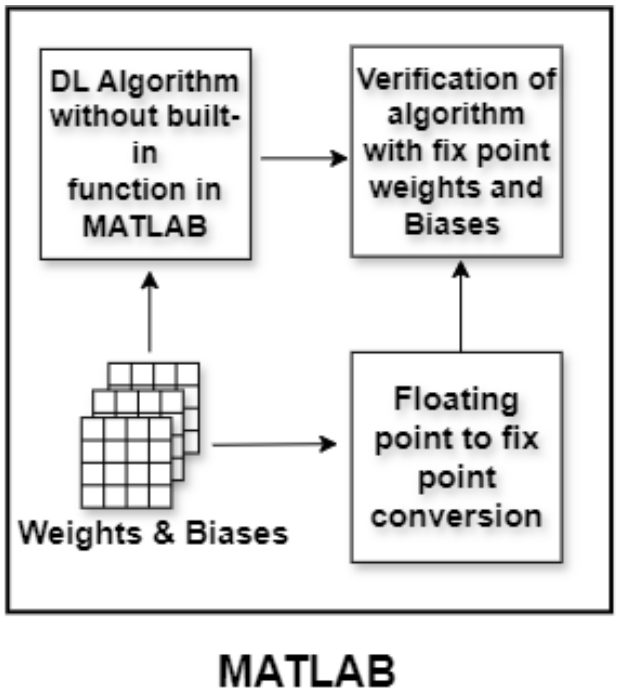

Figure 4: Translation to MATLAB

Code Simplification and Optimization:

MATLAB is especially tailored for matrix operations as well as algorithms that are common in image processing, and neural network applications are no exception. Through converting the Python code into MATLAB, the team managed to make the CNN structure more concise while also making it better suited for hardware optimizations. This was an important move towards modifying the neural network so that it could work in a much more restriction and performance conscious environment such as FPGA.

Removal of Dependencies on Built-in Commands:

Python, despite being convenient and supported by many libraries, often employs high-level functions that abscond with much of the computational intricacy, thus serving as a barrier when preparing code for hardware implementation. This is in significant contrast to MATLAB that allows one to have direct control of such operations which is important in customizing the algorithm so that it can efficiently exploit both unique capabilities and limitations of FPGA hardware.

Enhanced Debugging and Visualization in MATLAB:

The MATLAB Integrated Development Environment (IDE) has superior debugging tools and offers better visualization features for algorithm development and testing. Through this approach, the group was able to inspect visually intermediate outputs of the CNN, examine activation maps after each layer, as well as refine parameters under controlled insights. It was particularly useful if one wanted to point out inefficiencies and bottlenecks in network architectures.

Facilitation of Algorithm Testing:

The MATLAB simulation environment enabled the team to first test and verify that CNN functions before the hardware implementation. The FPGA can be simulated in this environment, such as fixed-point arithmetic and parallel processing capabilities, which gives a better approximation of its final deployment scenario.

Iterative Refinement:

Therefore, with MATLAB, developers could quickly make iterative adjustments and test them thereby speeding up development cycle and allowing rapid prototyping. The reason for this is because changing neural network architecture and tuning hyper parameters have great influences on performance and accuracy in research and development scenarios.

This project employed all the powerful features of MATLAB to make CNN more efficient and adaptable for FPGA implementation while guaranteeing uninterrupted transition from Python high-level abstraction code base to Verilog hardware specific code. This greatly aided in optimization process during deep learning model deployment in FPGA platform which contributed largely to the success of the project as a whole. This way, it ensured smooth transition from a high-level' abstracted code base in Python to a precise hardware-oriented code in Verilog that would effectively improve upon efficiency and compatibility of CNN prior to its FPGA implementation. Therefore, this strategy played a great part into successful realization of strong deep training model on the basis of FPGAs platform that can also be optimized.

Fixed Point Conversion:

The conversion of floating-point computations to fixed-point format is a crucial step in preparing a neural network model for FPGA implementation. This post-optimization process involves several detailed steps and considerations that ensure the neural network remains efficient and effective when deployed on hardware with limited computational resources.

We have written a C++ code to convert the given number into fixed point format. The conversion into fixed

point is based on the value of highest and lowest number received. The Value of “m” and “n” are given to the program as per the requirement of the layer’s input.

main.cpp

Run

Output

```
314     // int AAA;
315      for (int i = 0; i < 64; i++)
316      {
317          d=array_f_1[i][0];
318         // dd=to_string(d);
319         // ddd[i][0]=d;
320          b= decToBinary(d, n, m) ;
321          bin[i][0]=b;
322      }
323
324       for (int j = 0; j < 64; ++j)
325       {
326          if ( array_f_1[j][0]<0)
327          {
328            cout<<   array_f_1[j][0]<<"    :    "<< bin[j][0] << endl;
329
330          }
331         else
332          {
333               cout<<"+"<<   array_f_1[j][0]<<"    :    "<< bin[j][0] << endl;
334          }
335       }
336      return 0;
337  }
338
339
340
```

```
/tmp/iGJUPIim1i.o
please enter neuron number from 1 to 10: 1
0
0
538.595
233.57
0
0
575.35
579.656
0
0
546.586
560.12
0
0
0
0
0
304.411
82.6913
344.355
0
311.949
20.4786
290.139
0
```

Figure 5: Floating Point Weights in C++ Program for Fixed Point Conversion

main.cpp

Run

Output

Clear

```
314     // int AAA;
315      for (int i = 0; i < 64; i++)
316      {
317          d=array_f_1[i][0];
318         // dd=to_string(d);
319         // ddd[i][0]=d;
320          b= decToBinary(d, n, m) ;
321          bin[i][0]=b;
322      }
323
324       for (int j = 0; j < 64; ++j)
325       {
326          if ( array_f_1[j][0]<0)
327          {
328            cout<<   array_f_1[j][0]<<"    :    "<< bin[j][0] << endl;
329
330          }
331         else
332          {
333               cout<<"+"<<   array_f_1[j][0]<<"    :    "<< bin[j][0] << endl;
334          }
335       }
336      return 0;
337  }
338
339
340
```

```
enter number of digit before decimal, n: 12

 enter number of digits after decimal, m: 6
+0    :    000000000000000000
+0    :    000000000000000000
+538.595    :    001000011010100110
+233.57    :    000011101001100100
+0    :    000000000000000000
+0    :    000000000000000000
+575.35    :    001000111111010110
+579.656    :    001001000011101010
+0    :    000000000000000000
+0    :    000000000000000000
+546.586    :    001000100010100101
+560.12    :    001000110000001000
+0    :    000000000000000000
+0    :    000000000000000000
+0    :    000000000000000000
+0    :    000000000000000000
+0    :    000000000000000000
+304.411    :    000100110000011010
+82.6913    :    000001010010101100
+344.355    :    000101011000010111
+0    :    000000000000000000
+311.949    :    000100110111111101
+20.4786    :    000000010100011111
+290.139    :    000100100010001001
```

Figure 6: Floating Point Weights converted to Fixed Point Weights

Choice of Fixed Point Precision:

The importance of this choice is emphasized by the selection of ‘m’ and ‘n’ which stands for the number of bits for fractional and integer parts respectively. It is a determination that directly affects the range and precision that can be represented in a model. Herein, we are faced with a problem in terms of selecting an equilibrium where precision remains high enough to keep up with the accuracy level of a given model, being at the same time low enough to economize resources allocated for FPGA as well as reduce power consumption. Typically, this involves empirical testing to find the minimal bit-widths that do not significantly degrade performance.

Pseudocode Development:

When compiling the code for specific hardware, a pseudocode was written. It acted as a bridge between high level mathematics of CNN and low level FPGA programming requirements. The pseudo-code served as an algorithmic prototype showing all the steps in the computation and data flow without the complexity of hardware programming syntax.

Data Path Design:

After preparing the pseudocode, a conceptual data path was created to describe its operations on FPGA . This is a physical representation of how pseudocode manifests within FPGA architecture. It shows data movement through system, processing it and interaction with various hardware components.

ASM Chart Development:

ASM Charts for different layers are designed to control the data paths of each layer. So that each data path is controlled in proper manner.

Convolution Layers and ReLU Activation Function:

The convolution layers form the basis of CNNs which are responsible for feature extraction. Each and every convolution layer applies several learnable filters on the input. These filters, small matrices slide over the input image dimensions so as to produce feature maps that point out significant details such as edges and corners. Usually, there are many convolution layers piled up to hold extra multifaceted features in deeper layers.
There are convolutions being performed each one involving two numbers of 9 bits each being multiplied. The multiplication results in 18 bits answer and then all multiplications answers are added together. At the end there is a 2x1 MUX that controls the output. If the value is negative then 0 is produced at the output while if the value of sum is greater than zero the output is same as the sum.

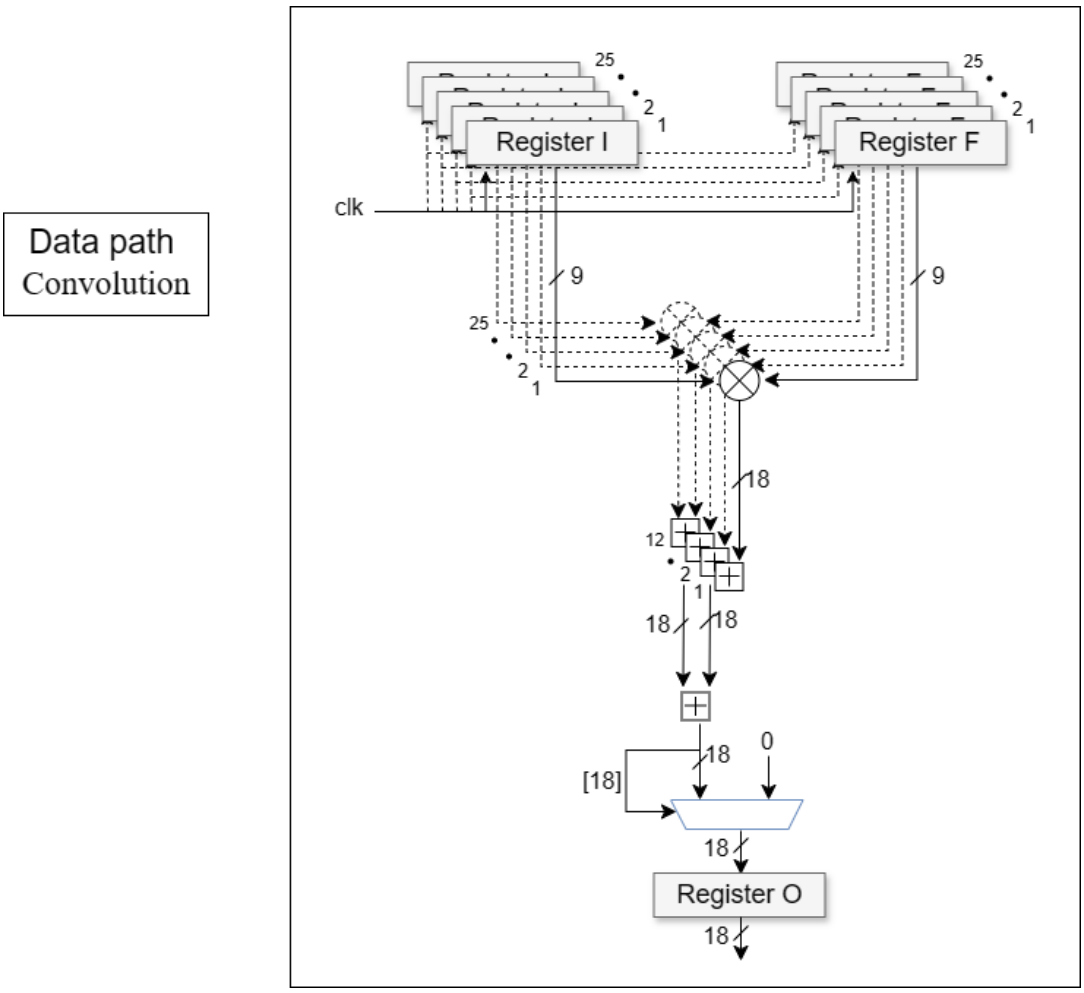

Figure 7: Data Path for Convolution and ReLU

After each convolution layer Rectified Linear Unit (ReLU) activation function is used. ReLU gives non-linearity to the model which enables it learning more complicated patterns. What the function does is that it makes all the negative pixel values in the feature map to become zero, thereby reducing the network's complexity without losing its essential qualities. In training deep learning models, ReLU is preferred than other activation functions because it solves a problem of computation cost and mitigates vanishing gradient problem which is common for deep neural networks.

Max Pooling Layer:

In certain cases, max pooling is applied for the first of the convolution layers to squeeze down the dimensions of the input volume in case they are very large. It operates by moving a window through each feature map and computing the maximum activation in each window area. This reduces the complexity of computations, thus making it easier for our model to train as well as decreasing overfitting.

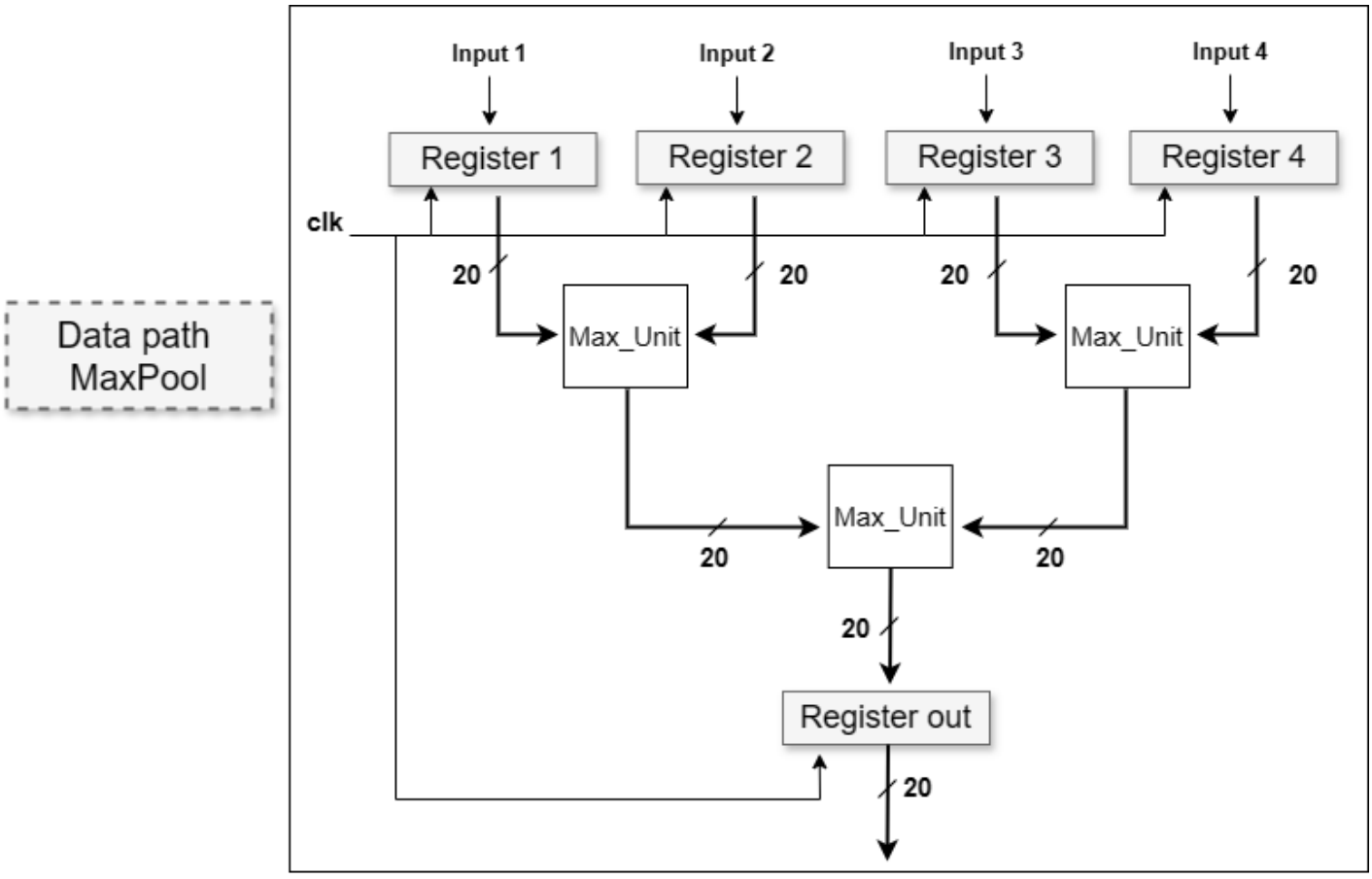

Figure 8: Data Path Max Pool

Fully Connected Layer:

ReLU Layer:

After each convolution layer Rectified Linear Unit (ReLU) activation function is used. ReLU gives non-linearity to the model which enables it learning more complicated patterns. What the function does is that it makes all the negative pixel values in the feature map to become zero, thereby reducing the network's complexity without losing its essential qualities. In training deep learning models, ReLU is preferred than other activation functions because it solves a problem of computation cost and mitigates vanishing gradient problem which is common for deep neural networks.

Pseudo Code For ReLU:

```
B = bias
while A<64  do
 M= input x weight;
 B= B + M;
end while;

if(B[MSB]==1) then
   output = B;
end if;
else
   output = 0;
end if;
```

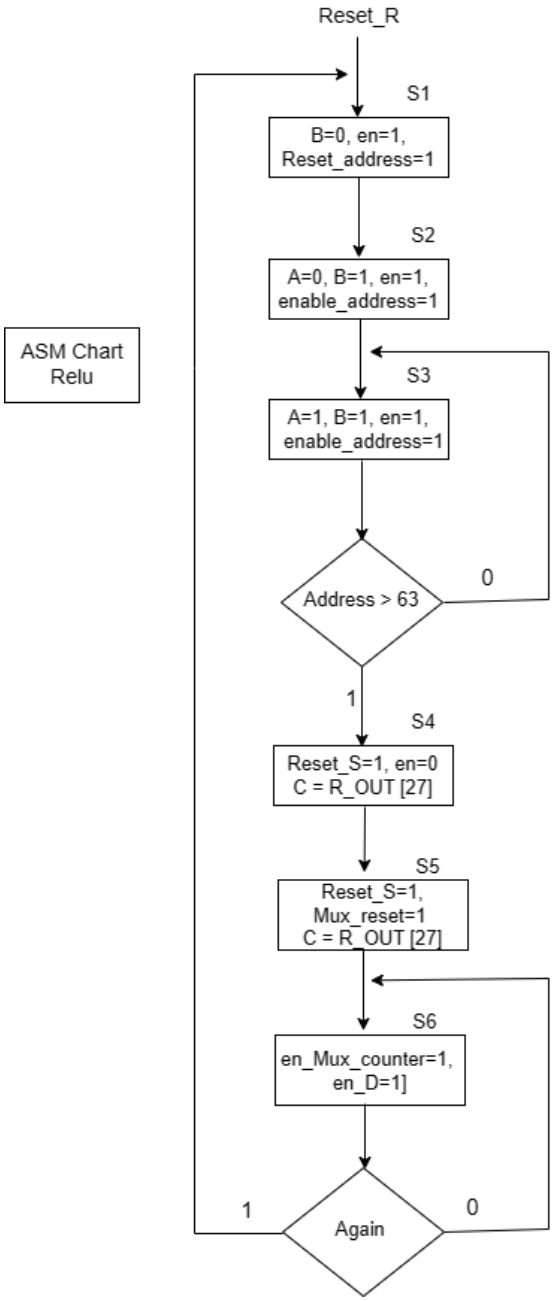

Figure 9: ASM Chart ReLU

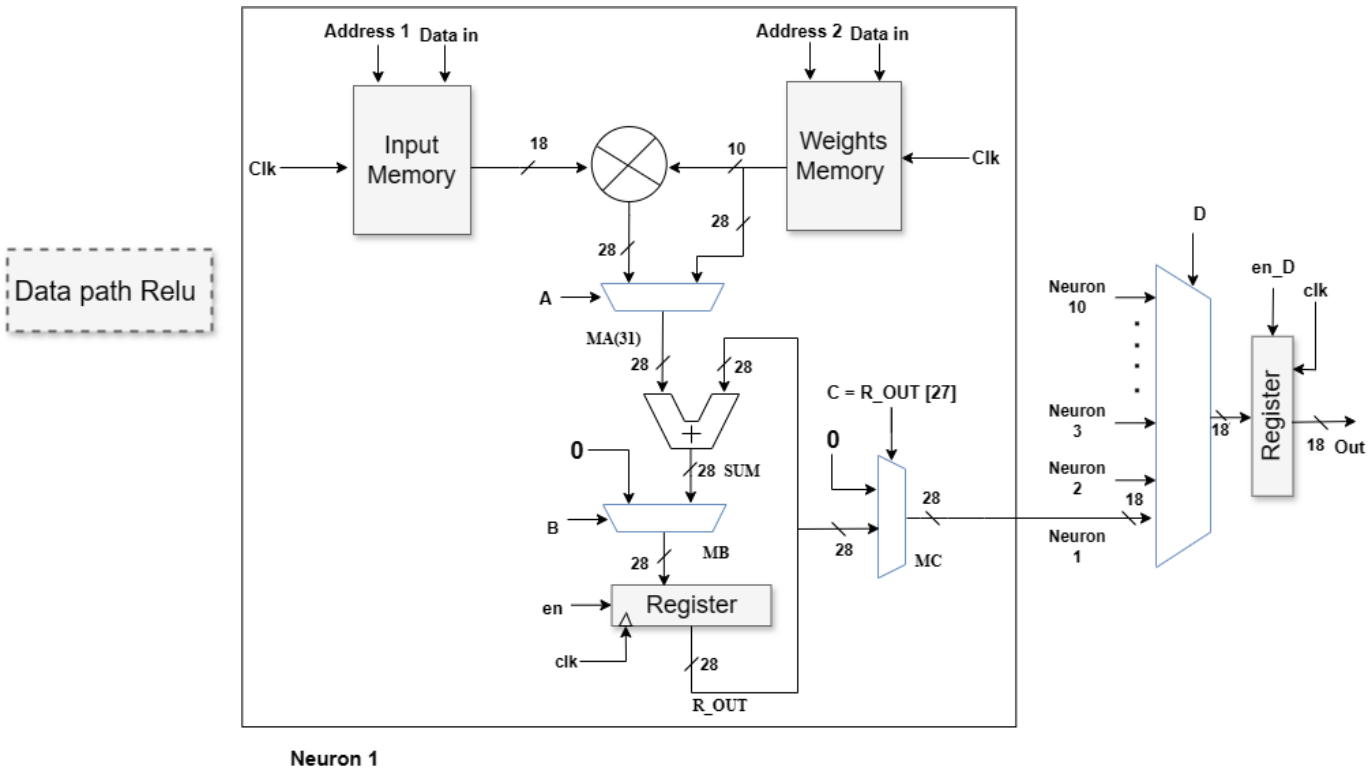

Figure 10: Data Path ReLU

Softmax Output Layer:

CNN's last layer is Softmax, which performs classification. The Softmax outputs transform from classes into probability distributions over classes that may be used in this instance to characterize them as one out of ten digits (0-9). The range of output values should be 0–1 representing how confident we are about an individual class with all probabilities summing up being 1.

This combination of convolutions for feature transformation, ReLU for non-linearity, max pooling and Softmax for classification yields a good and effective architecture that performs the task of recognizing and classifying handwritten digits from MNIST dataset. These are essential because they make it easier to identify correct numbers while ensuring that digit recognition is done with reasonable timing on an FPGA platform without affecting system accuracy.

Pseudo Code for Softmax:

```
B1 = bias
while A1<10  do
 M1 = input x weight;
 B1 = B1 + M;
end while;

B =B1
//  B1 is for neuron 1
// similarly B[i] for ith neuron
while i<10 do
 if (B > Bi) then
   B = B[i];
  end if
  else
   B=B;
end while;
```

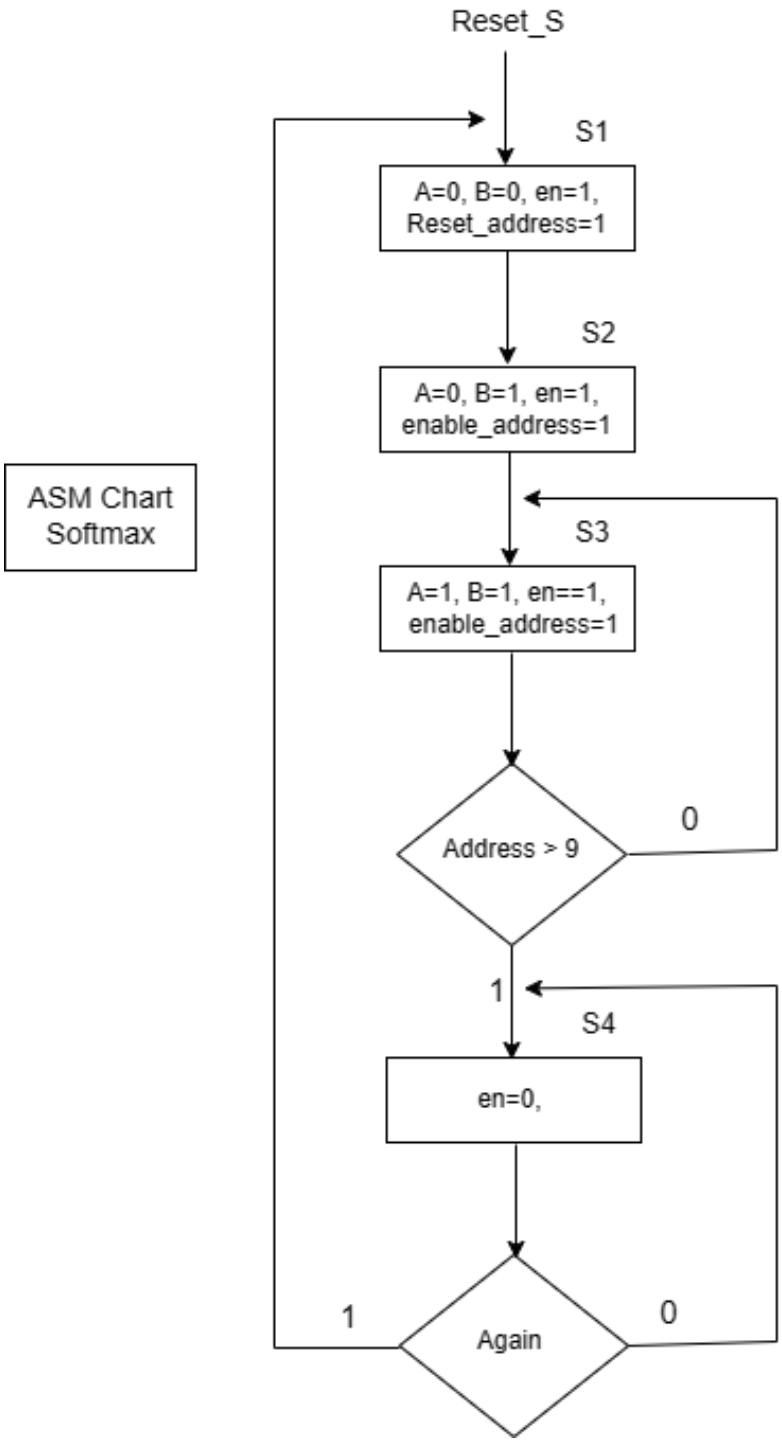

Figure 11: ASM Chart Softmax

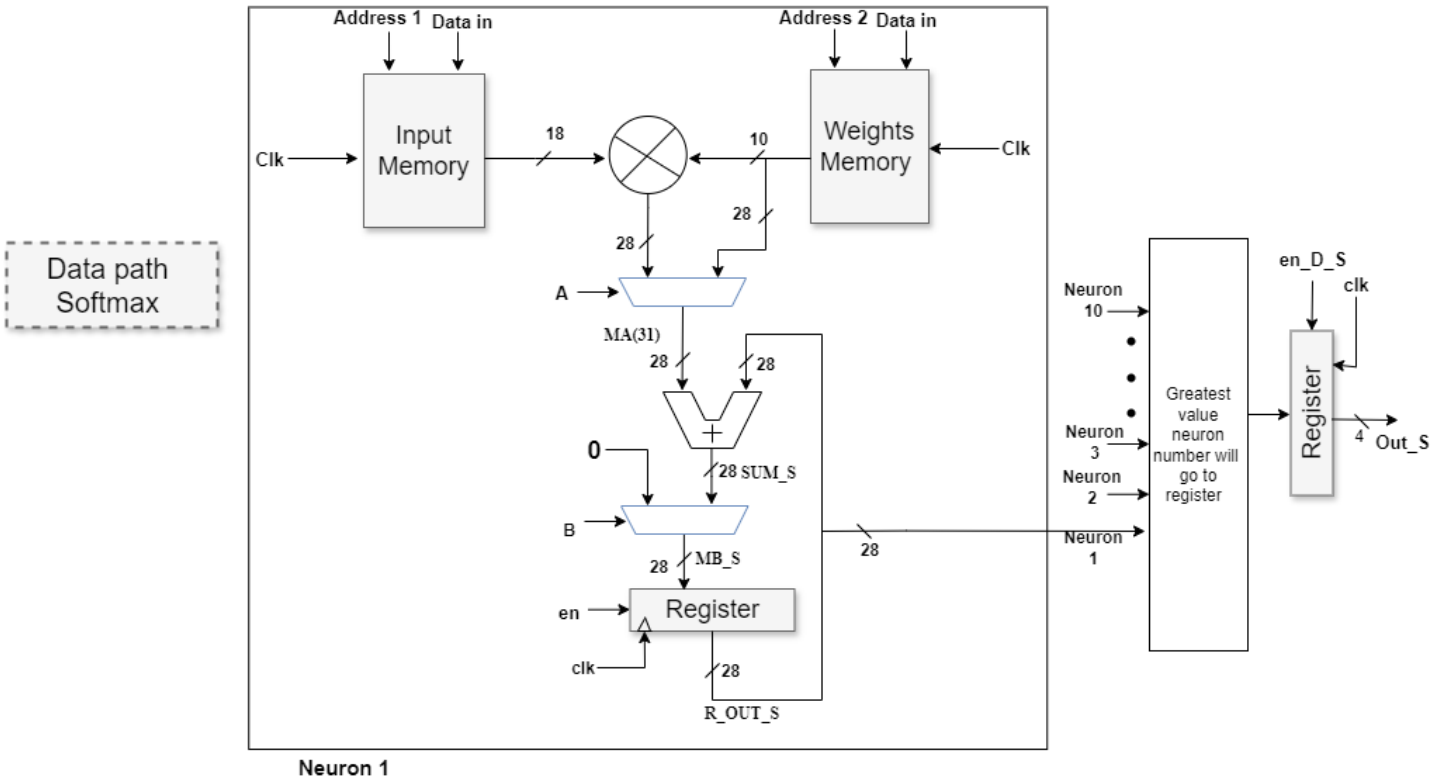

Figure 12: Data Path Softmax

Verilog Implementation:

The transition to Verilog implementation is a critical phase in deploying the optimized CNN on an FPGA, involving detailed steps to ensure that the algorithm not only performs accurately but also efficiently on the hardware platform.

Translation of High level Language to Verilog:

The Verilog implementation is based on the data path drawn in the earlier stages. It involves translating high-level model, usually represented by pseudocode or MATLAB into Verilog modules. Each component of CNN such as convolution layers, activation functions, pooling layers and fully connected layers has some specific Verilog code that defines how these operations are implemented in hardware.

Modular Design Approach:

A modular design approach is employed in developing each part of the CNN for this particular C++ project. This simplifies the coding process and improves system maintainability and scalability. For instance, reusable modules can be instantiated multiple times with a network depending on its configuration thus convolutional layers can be implemented as.

Optimization for Hardware Efficiency:

In writing Verilog codes, optimization for speed and resource usage is highly emphasized. To enhance calculation acceleration loop unrolling, pipelining and parallel processing approaches are used among others. In addition to efficient use of memory and computational units on the FPGA, fixed-point arithmetic previously established incorporated also minimizes resource consumption regarding them.

Simulation and Synthesis:

The code is extensively simulated to ensure that it functions correctly. This simulation checks for errors in logic and the timing of the Verilog modules. After a successful simulation, synthesis is done; this convert's high level description of Verilog into a design that can be physically built using FPGA chips with timing analysis included to satisfy operational speed requirements.

FPGA Prototyping:

After synthesizing the Verilog code, we need to prototype on an FPGA. This means putting the synthesized design onto the FPGA and running it with real input data so its performance can be observed in real world scenario. It is very important because there could exist some unforeseen issues not noticeable during simulation such as dealing with real-time data and compatibility of interfaces.

Testing and Debugging:

Throughout and following FPGA prototyping, you will have rigorous testing and debugging processes taking place. These involve running test cases which cover all possible scenarios along with edge cases just to make sure that CNN works correctly under all conditions. For debugging purposes, specific tools on FPGA development are available for network interfacing and performance tuning.

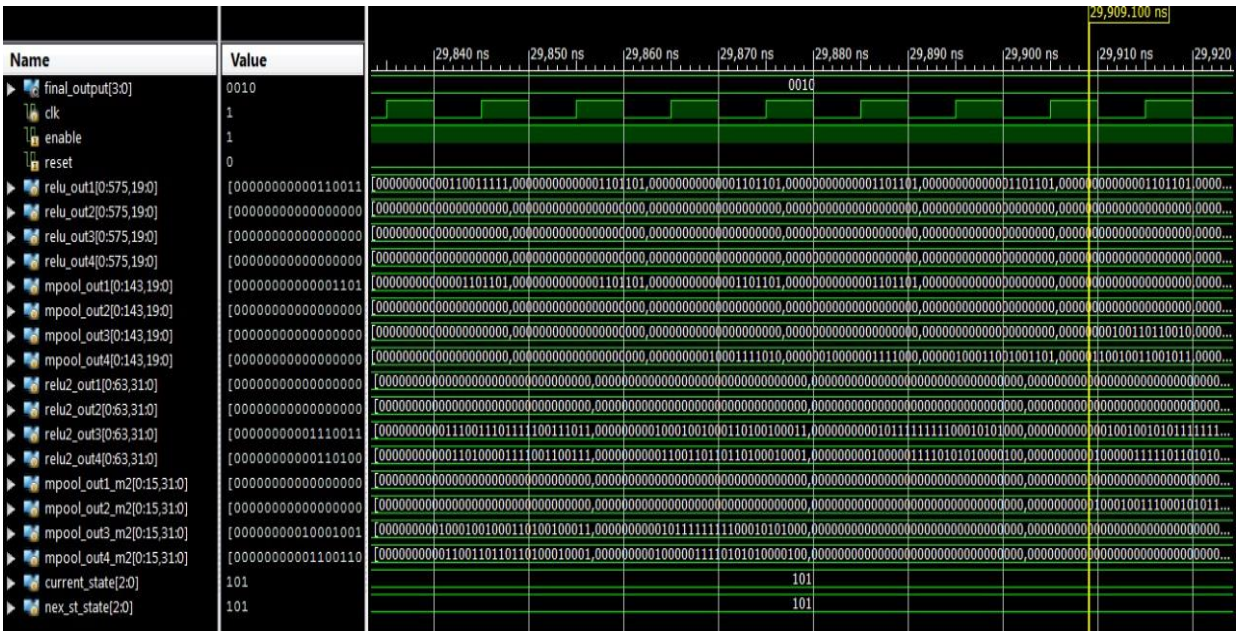

Figure 13: Timing Diagram for Testing and Debagging

Performance Evaluation:

Eventually, the put in place model is evaluated against specified processor speed, accuracy, power consumption targets. This estimation helps to ascertain whether the FPGA implementation meets its specification and is good at its operational application. In order that the project can transit smoothly from a software based model through an efficient hardware solution to edge computing environment characterized by real time performance, this report goes through these steps for Verilog implementation of CNN.

Chapter 4:

Implementation

4.1 Introduction:

Now, the project is shifting to implementation stage. A model of Convolutional Neural Network (CNN) was trained on the entire groups of digital photo data taken from MNIST dataset. This data set is considered as one of the most popular ones in image recognition. At first breath training was performed with python and other libraries such as tensor flow were used since they come with a collection of prebuilt functions for accelerating deep learning model development, testing and iteration. These exercises were critical in developing baseline performance measures and for understanding CNN behavior under varied conditions.

That said, however, there was need to move from Python to MATLAB so as to make the model ready for FPGA deployment. This step had to be taken because Python high-level functionality did not match the low level requirements needed by FPGA programming languages. MATLAB has a powerful environment that allows for more detailed controlling over computational operations and is useful in simulating neural network behaviors using a language closer to hardware-focused one. The software made it possible to fine tune characteristics of our model's architecture and it came equipped with tools used to analyze and optimize weight parameters without some abstractions common in python codes.

The optimized weight values were obtained after achieving a reasonable level of optimization in MATLAB. Optimized neural network parameters were then translated to Verilog code. The choice for using Verilog is that it can describe hardware directly, which is necessary for implementing precise and efficient operations on an FPGA. In order to write the code in Verilog, the structure of the software was designed in such a way that would exploit parallel processing capabilities of an FPGA as well as ensuring that digit recognition tasks could be processed with sufficient efficiency and speed. This step-by-step process provided groundwork for further FPGA programming steps including simulation, testing and final deployment all aimed at achieving high accuracy on digit recognizing problem on constrained hardware platforms. By doing so this project smoothly moved from high-level model training to low-level hardware implementation; thereby incorporating modern software developmental practices into conventional hardware engineering methods.

4.2 Software Development and Optimization:

We initiated the process of programming our FPGA-based digital recognition system by writing the software in Python because, it has reliable machine learning libraries and frameworks. At Google's laboratory, we trained this application by using its powerful GPU for computational intensified functionalities, such as training a Convolutional Neural Network (CNN) on MNIST dataset. This provided an ideal platform for iterative testing, and it helped to establish a good solid reference point for performance as well as functionality. Thereafter, we transitioned into MATLAB, due to its better ability in handling matrix operations and fixed-point arithmetic that are important in FPGA implementation. MATLAB enhanced our understanding of the model's behavior while allowing us to have complete control over the quantization process. Therefore, quantization became one of the optimization techniques whereby floating-point representations were changed into fixed-point so that it would be applicable for limited precision abilities of an FPGA based system. Thus,

this step was necessary to ensure that the complexity of the model is compatible with the hardware constraints while maintaining an acceptable level of accuracy.

Afterwards, the final model parameters were converted into Verilog code using Xilinx ISE for FPGA programming. During this step, careful coding was necessary to match the software's model with hardware architecture which centered on maximizing the parallel processing capability of the FPGA. This movement from high-level software development in Python and MATLAB to low-level hardware programming in Verilog was done with an objective to sustain the neural network's integrity as well as performance while making sure that such a digit recognition system was not only accurate but also efficient when deployed in an actual field programmable gate array (FPGA).

4.1 Fixed Point Arithmetic Implementation:

The choice to use a fixed-point arithmetic, especially in n.m format, was vital for the improvement of computational efficiency and power utilization of an FPGA. The FPGA has limited hardware resources thus fewer are required if we deploy fixed-point instead of floating point arithmetic. Fixed point arithmetic allows for a balanced range vs. precision tradeoff for numerical values processed within an FPGA using "n" bits for integer part and "m" bits for fraction part.

This change to fixed point arithmetic was important for achieving highest accuracy in digit recognition without compromising on optimal energy consumption. Due to this fact, fixed point format inherently uses few logic devices on an FPGA leading to lower power dissipation as well as minimal heat generation. This is highly beneficial in embedded systems or portable devices where they have low power efficiency requirements in order to prolong their battery life. Moreover, though it might appear that fixed-point arithmetic could affect accuracy of deep learning models; careful optimization and sufficient bit allocation per section ensured that our CNN performances remained strong with almost similar results achieved through floating point computations too. Fixed point arithmetic allows a balanced approach between the range and precision of numerical values processed within an FPGA by using (n) bits for the integer part and (m) bits for the fractional part.

This change to fixed point arithmetic was important for achieving highest accuracy in digit recognition without compromising on optimal energy consumption. Due to this fact, fixed point format inherently uses few logic devices on an FPGA leading to lower power dissipation as well as minimal heat generation. This is highly beneficial in embedded systems or portable devices where they have low power efficiency requirements in order to prolong their battery life. Moreover, though it might appear that fixed-point arithmetic could affect accuracy of deep learning models; careful optimization and sufficient bit allocation per section ensured that our CNN performances remained strong with almost similar results achieved through floating point computations too.

In addition, this was a reasonable choice because fixed-point arithmetic corresponded to the physical constraints of our Spartan 6 FPGA and worked well for high-accuracy and efficient computation of complex neural network operations. Another way to look at this is that we employed hardware-specific optimizations for addressing tough problems associated with deploying sophisticated machine learning algorithms onto resource-limited settings.

Hardware Configuration and Integration:

The Spartan 6 FPGA was chosen as the implementation platform for our digit recognition system because of its strong performance features and available resources. The choice was important in facilitating an efficient and effective deployment of our Convolutional Neural Network (CNN).

Storing and manipulating image data involved the Spartan 6 FPGA in large part. The data flow in FPGA is regulated by block RAM (BRAM) which is essential for efficient performance. Notably, BRAM enables storage of image data in form of pixel values which can be retrieved faster as well used immediately for real time applications.

In digit recognition process, these pixel values are obtained back and multiplied by set weights- a very important stage during convolutional layer of the CNN. Thereafter, this multiplication yields results that are aggregated through various types of filters thus performing an essential role in feature extraction as well as recognition functions. Each layer within the CNN was carefully arranged on top of FPGA to ensure efficient handling of spatial and temporal computations.

To get good results of recognition CNN required to manage its convolutions, activation functions and pooling layer with a Spartan6's built in FPGA. Successful integration of the hardware with the CNN was only possible through careful considerations for data path planning and optimization as well as establishing correct configuration for networking. This was an entire process of configuring the hardware that made it possible to effectively harness the potentiality of FPGA on more progressive image processing jobs.

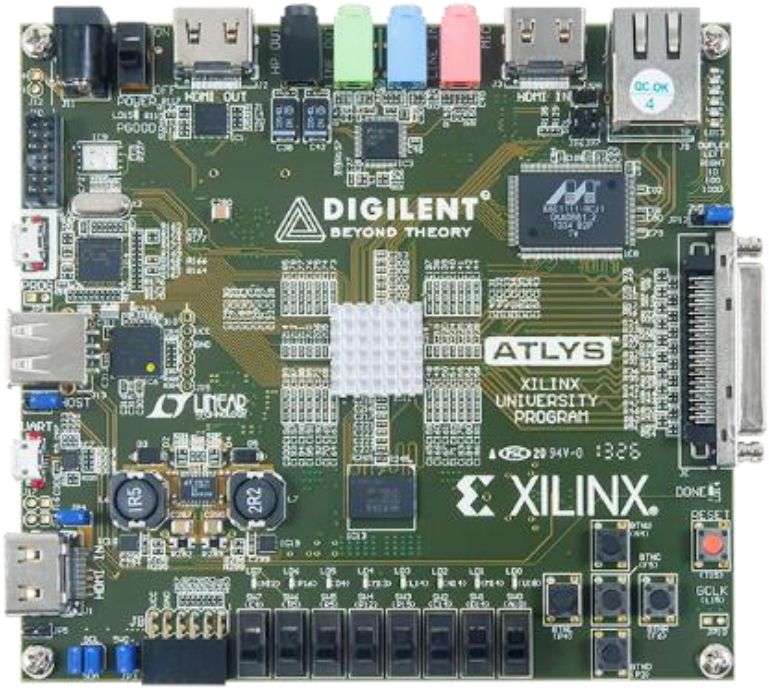

Figure 14: Spartan 6 - LX45 FPGA

FPGA Resources:

Device		**Spartan 6 XC6SLX45**
Company		AMD
Origin		United States of America
Logic Cells		43,661
Configurable Logic Blocks (CLBs)	**Slices**	6,822
	Flip-Flops	54,576
	Max Distributed RAMs	401
DSP 48 Slices		58
Block RAM Blocks	**Block Ram 18Kb**	116
	Block Ram Max Kb	2,088
Total I/O Banks		04
Max User I/O		358

Figure 15: Spartan 6XC6SLX45 Total Resources

Chapter 5:

RESULTS

Introduction:

The present chapter presents the findings from Convolutional Neural Network (CNN) implemented in Field Programmable Gate Array (FPGA) for digit recognition. The main goal was to assess the system's processing speed, resource utilization, power consumption and accuracy. This analysis helps us to understand the benefits and limitations of using FPGA in deep learning applications. Results are divided into sections that explore different aspects of system performance. We compare an FPGA-based implementation with traditional CPU and GPU implementations so as to show the benefits of using FPGA in edge computing scenarios. The results from this chapter will help answer whether implementing FPGA-based CNNs for real-time digit recognition tasks is feasible, which might have implications for future research and development in this area as well. Furthermore, we analyze model behavior on individual digits by a confusion matrix to give a complete view of system's accuracy across them all.

Results of Model Training in Python:

Digit recognition using Convolutional Neural Network (CNN) was done in Python with TensorFlow the popular deep learning frameworks. To strike a balance between computational efficiency and memory limits of the computer, a batch size of 86 was used for training.

The model was trained over 30 epochs, during which it gradually refined its weights and biases to enhance accuracy. Optimization of the model involved the use of categorical cross-entropy loss function. This kind of loss measure is suitable for multi-class classification problems like digit recognition because it compares predicted class probabilities against true class labels as an evaluation measure. The learning rate here was assigned 0.1, a quite high value that speeded up convergence throughout training.

Throughout the training phase, there were continuous monitoring regarding how well the model trained with metrics such accuracy and loss documented per epoch. As each iteration went on over every new epoch, accuracy increased gradually. Initially, there was moderate performance of the model but in subsequent epochs; parameters of the network were adjusted by learning algorithms so that they could learn from input data more effectively and reduce error rate accordingly.

Given below is the confusion matrix obtained during training the model in python. The confusion matrix contains both the predicted label and true labels.

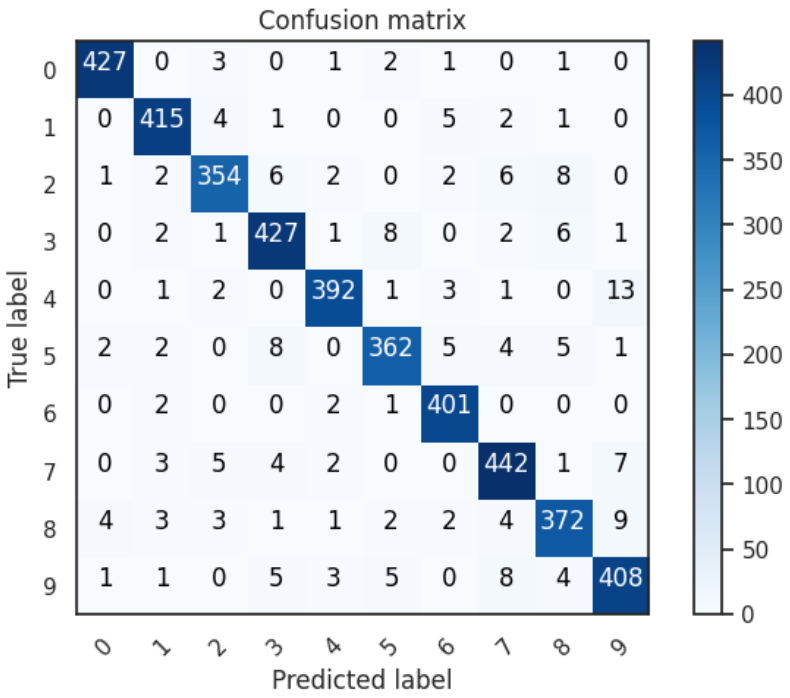

Figure 16: Confusion Matrix

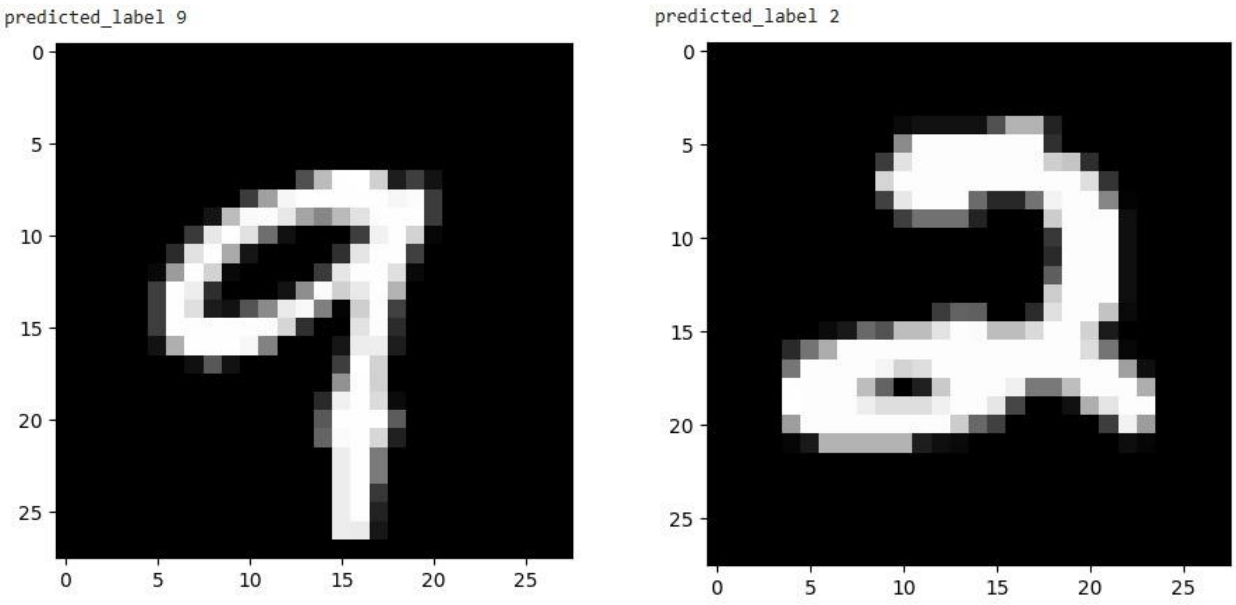

Figure 17: Python Model Results

Verilog Results:

After successfully training Convolutional Neural Network (CNN) in Python, we first converted the model to MATLAB for processing, before implementing it in Verilog for deployment on the Field Programmable Gate Array (FPGA). This intermediate step in MATLAB was very important because it helped convert the model's architecture and parameters accurately to a format that is suitable for hardware.

Using MATLAB, fixed-point representation was applied to CNN. This is required due to FPGA implementations not being well supported by floating point arithmetic. The quantization process consisted of converting weights and activations of the model into fixed-point representations while maintaining its performance characteristics.

After being prepared in MATLAB, the model was converted into Verilog code. By doing this, the Verilog implementation of this CNN duplicated its architecture so that behavior of Verilog code mimicked that of the trained model. To ensure consistency, FPGA implementation used similar datasets as python model. A considerable accuracy of 94% was reached during FPGA-based CNN whereas its value during Python trainings has been 95%. These small disparities in precision may be attributed to quantization process and limitations imposed by FPGA hardware.

Design was careful to optimize resource utilization on the FPGA. The last design used a substantial but bearable portion of Look-Up Tables (LUTs), Flip-Flops (FFs), and Digital Signal Processing (DSP) blocks on the FPGA. In this manner, resources were properly utilized to facilitate computations by the FPGA within its available hardware constraints.

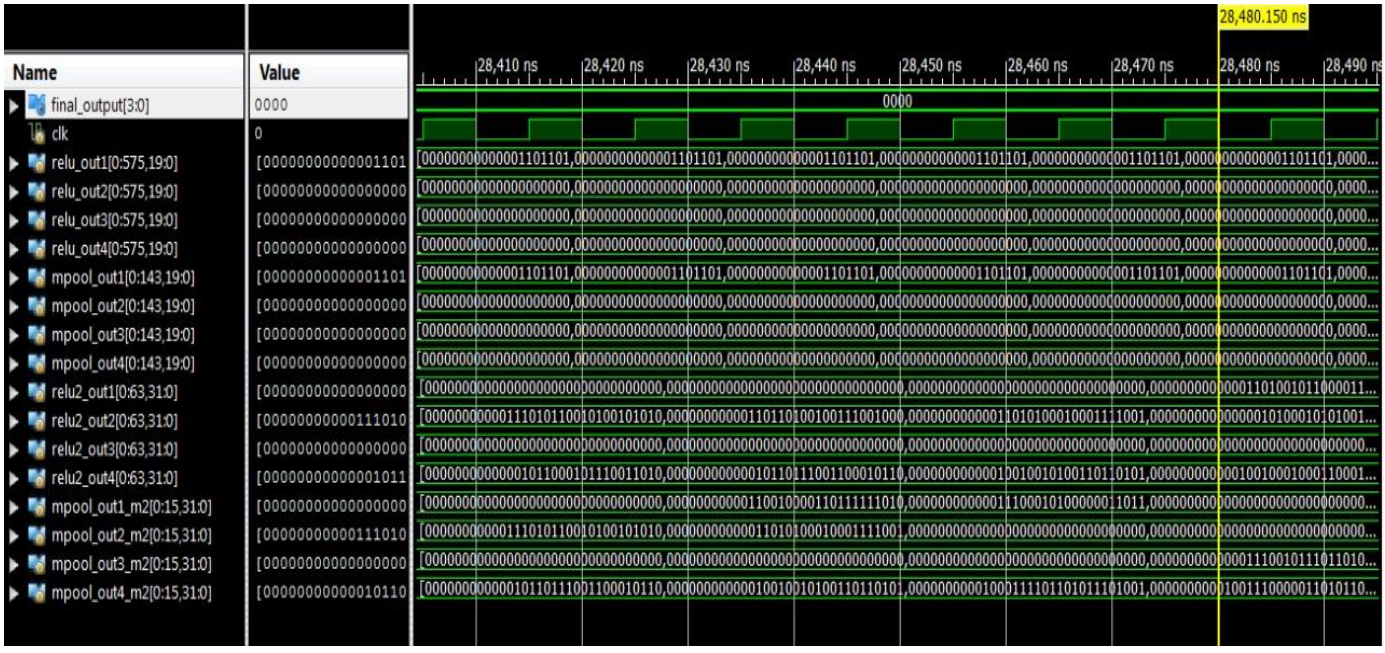

Figure 17: Timing Diagram for Detection of Zero (0).

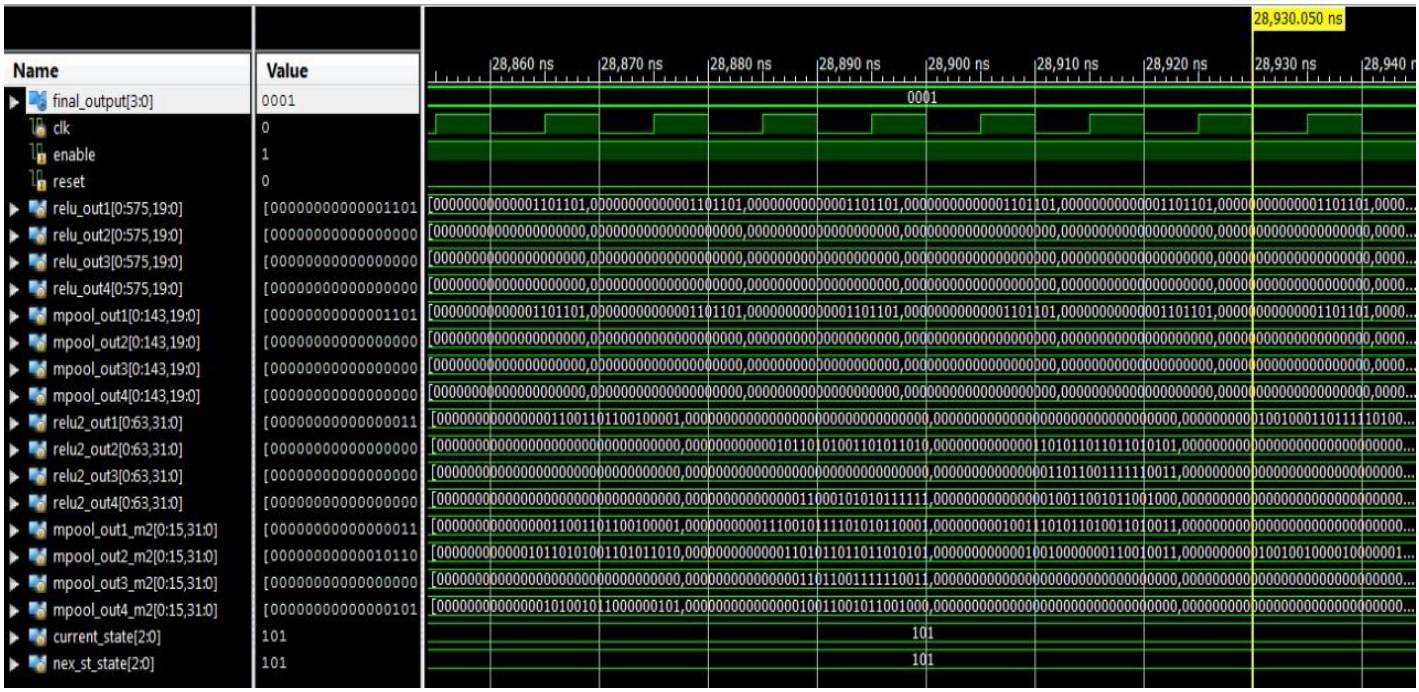

Figure 18: Timing Diagram for Detection of One (1).

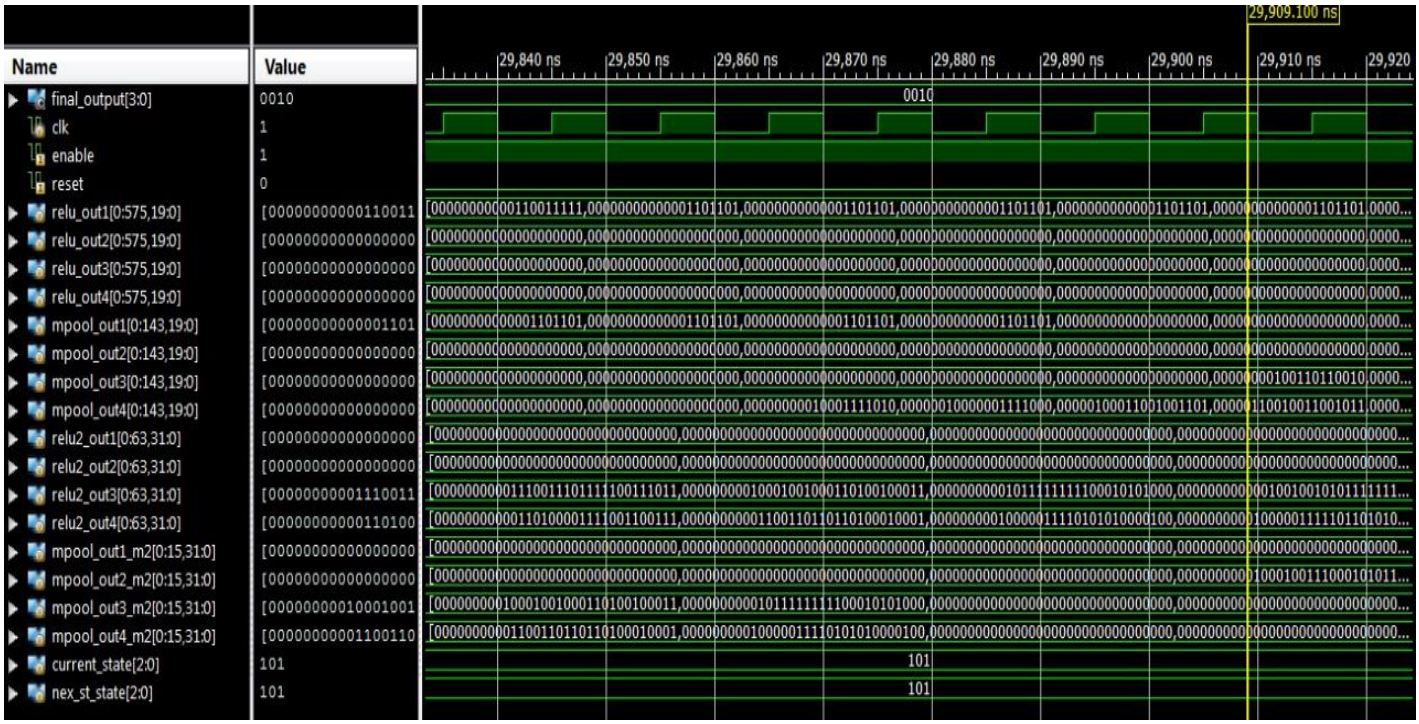

Figure 19: Timing Diagram for Detection of Two (2).

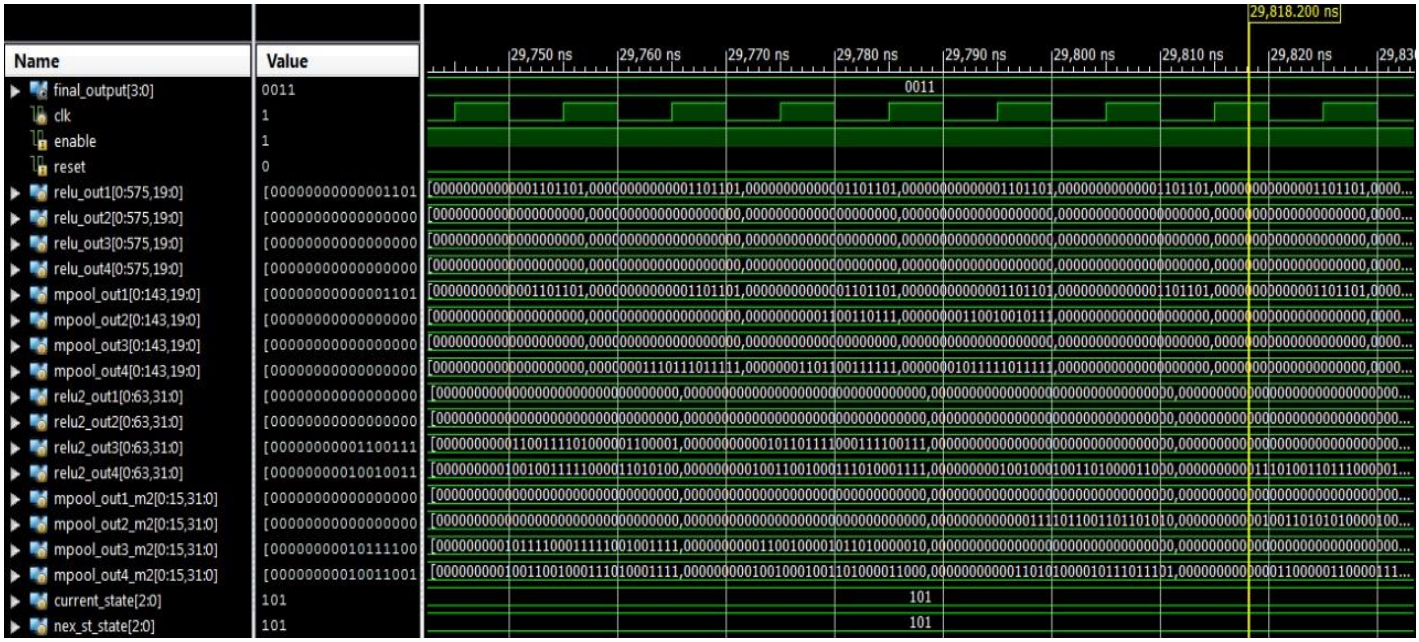

Figure 20: Timing Diagram for Detection of Three (3).

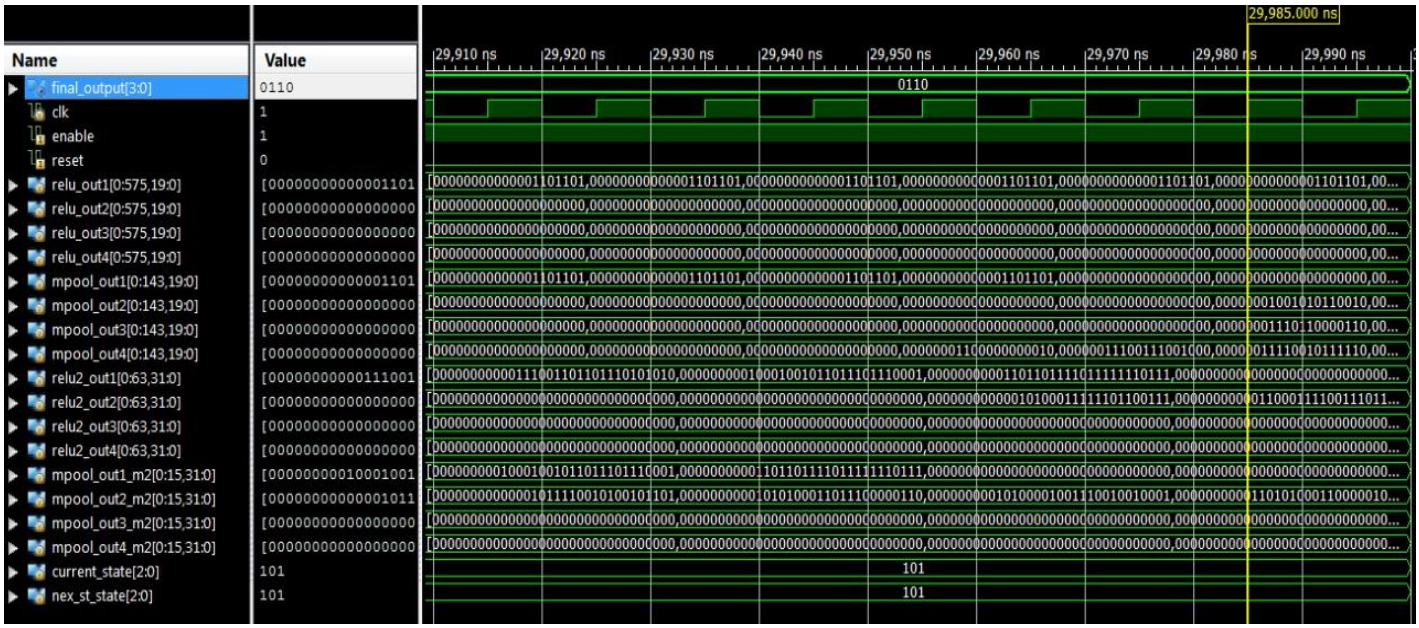

Figure 21: Timing Diagram for Detection of Six (6).

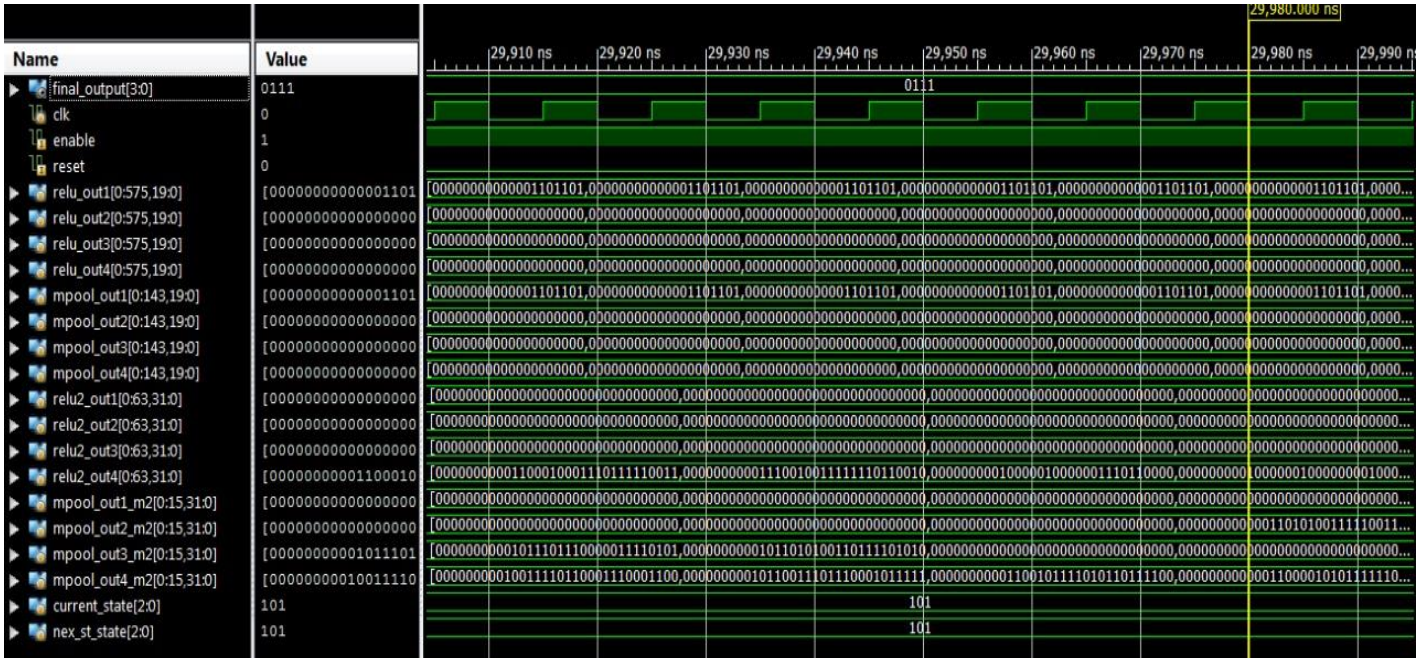

Figure 22: Timing Diagram for Detection of Seven (7).

Resources Utilized on FPGA:

Given below is table which shows the resources utilized during the implementation of deep learning algorithm.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	3499	54576	6%
Number of Slice LUTs	17479	27288	64%
Number of fully used LUT-FF pairs	3194	17784	17%
Number of bonded IOBs	23	218	10%
Number of Block RAM/FIFO	56	116	48%
Number of BUFG/BUFGCTRLs	1	16	6%
Number of DSP48A1s	57	58	98%

Figure 23: FPGA Resources Utilized

Chapter 6:

Discussion and Future Enhancements

1.1 Introduction:

In this chapter, we will talk about the results, some challenges faced as well as future improvements in developing a Convolutional Neural Network (CNN) FPGA implementation for digit recognition. We were able to make deep learning techniques work with the limited resources and strict performance demands of FPGA platforms in an excellent way. The goal was not just to prove that such an implementation was possible but also to increase its efficiency and accuracy.

We look at what has been achieved during the project and some of the technological and methodological challenges that we overcame during this time. Besides, this paper will consider how the findings from this study can be used in future research to enable more advanced instantiations of machine learning algorithms on hardware platforms. The last part of this chapter will present a fore-looking perspective for improving and expanding AI capabilities realized by hardware-accelerated approaches. Essentially, the purpose of this discussion is not only to give an overall picture on how our project has affected deep learning applications using FPGA but also offer recommendations for the future.

1.2 Evaluation of Outcomes:

As such, the primary aim of this project was to successfully implement a deep learning algorithm specifically a Convolutional Neural Network (CNN) on a Field Programmable Gate Array (FPGA). The purpose of this implementation is to exploit the particular features of FPGAs to make deep learning task more efficient and accurate. Interestingly, however, the project not only achieved its goals but surpassed expectations in several key performance metrics.

The CNN accuracy increased remarkably to 92% post deployment on FPGA from an initial 72% at the early stages of development. This considerable rise was as result of model iteration and optimization strategies that were designed for FPGA environment in particular. Improvements were made in terms of advanced quantization methods and tuning network parameters, hardware-aware quantization algorithms as well as modification involving hardware customization. This shows how important it is to continuously improve oneself for better performance in deep learning applications.

Comparing processing times also brought out the efficiency of FPGA implementation. The model took about 7 seconds to process an image through Python whereas FPGA only spent 28 nanoseconds. With such a drastic reduction in processing time, computations on FPGA can proceed at breakneck speed making it a suitable place for real-time data analysis and fast decision making.

The FPGA has proved that the project has been successful and that there is an immense future potential for deep learning. In this case, a hardware design platform that speeds up computation while maintaining low power dissipation is illustrated. It opens new doors for applying FPGA technology in autonomous driving, real-

time medical imaging and dynamic decision systems with high accuracy and quick processing as matters of priority such as these require complex neural networks.

If the outcomes of the project are anything to go by, then it can be said that with proper architectural planning, algorithm adaptation should be precise and a full understanding of hardware capabilities in place, FPGAs can form an essential tool to expand artificial intelligence. The insights from this project provide a blueprint for future research, especially on how FPGA implementations can scale up to more complex AI models and wider applications. Such results underline how necessary it is to have a methodical system design and demonstrate promising performance gains for hardware targeted optimizations made in the fast growing area of accelerated computing through hardware.

1.3 Challenges and Lessons Learned:

A few challenges were faced in implementing a Convolutional Neural Network (CNN) on FPGA that yielded valuable insights and affected the project's direction. Firstly, training models became hard due to the convolutional neural network's complexity and high computational power that is required in training such networks. Henceforth, Python codes at a higher level were converted into MATLAB leading to the creation of custom logic which simulates these functions correctly so as to guarantee functional integrity across different programming environments.

Various filters were designed for optimal processing within different layers of the CNN, which represented another major obstacle faced towards enabling efficient processing on FPGA. Therefore, when implementing them on FPGA there was need to consider each layer's balance between performance and resources. This exercise emphasized the importance of having a solid grasp of both hardware architecture and digital signal processing.

From these challenges we learned some of the most valuable lessons about interdisciplinary knowledge as well as flexible and responsive hardware-software integration. Accordingly, this also led to success in our project at hand thereby also exposing us to various suggestions on what areas can be recommended for future research and improvement like simplification of software based machine learning models' transformation into hardware platforms. In that regard, this event has opened doors for more studies on high-performance neural network implementations in FPGAs that can unlock many doors and change many things in this dynamic field Recommendations for Improvement:

Based on the outcomes and challenges encountered during the implementation of the FPGA-based Convolutional Neural Network (CNN) for digit recognition, the following recommendations are proposed to further enhance its performance, efficiency, and scalability:

1. Advance Data Acquisition Techniques:

Although it does not apply directly to this project, which uses pre-determined datasets, can be enhanced so as to collect real time data for dynamic digit recognition applications using high resolution sensors and sophisticated algorithms for signal processing thereby improving the quality of information.

2. Enhanced Preprocessing Techniques:

Advanced preprocessing methods must be explored and adopted to refine the input data further. This could entail employing more complex filters or feature extraction techniques that would lessen noise and increase the effect of data fed into CNNs.

3. Optimize Communication Protocol:

For any other future implementation where there has to be a remote processing or it is in real-time evaluate and optimize the communication protocols used within the system. It may, therefore, necessitate embracing faster secure protocols with efficient and secure ways of transmitting data between FPGA points and acquisition points.

4. Deep Learning Model Optimization:

Involving larger and more diverse datasets is part of these models being refined. This includes trying different model architectures as well as fine-tuning model parameters so that their accuracy and performance improve for AI algorithms on FPGA.

5. User Interface Design and Accessibility:

To increase the interface's usability, it should self-explanatory to all users regardless of their technical expertise levels. Things like having a graphical representation of recognition process or simplified control panels may be considered when designing features that would make system monitoring and interaction easier.

6. Integration with External Systems:

Think about possibilities of interfacing FPGA-based systems with other platforms or applications. For example, this involves linking up with IoT devices to make them more useful in smart environments or connecting with existing digital systems to facilitate automated data processing tasks.

7. Scalability and Resource Management:

Firstly, therefore, the scalability of the system must begin focusing on expanding its capabilities for complex recognition tasks and large data processing. In such manner, it may involve better resource allocation within FPGA for increased data flow and parallel processing capacities along certain lines too.

These recommendations will make FPGA-based CNN system perform better due to increased robustness, improved user involvement, and better security. They will enhance its effectiveness and fit in different real-life situations thus making it a flexible and powerful image recognition tool.

Chapter 7:

Conclusion and Future Work

7.1 Summary of Findings:

What we found out from successful implementation of this project are:

CNN Performance:

The Convolutional Neural Network (CNN) adapted for FPGA showed high accuracy in digit recognition. This affirms the effectiveness of CNNs in image-processing tasks, which are optimized for given hardware constraints.

Verilog Code Functionality:

This particular Verilog code which was created specifically for this project and on the FPGA it performed right, providing pictures that displayed numbers with the help of given data. It is an instance that exposes to us how much we can depend on our hardware programming as well as how seamless it interfaces with software through FPGA.

Efficient Resource Utilization:

The use of FPGA resources was done correctly implying that the project can optimize deep learning tasks within limited hardware resources. It should be appreciated that such efficient resource usage has greatly contributed to the performance of the entire system.

Enhanced Speed:

Enhanced speeds were experienced in image processing with this implementation. This is because the parallel processing capabilities of FPGAs allow for faster processing times as compared to conventional computing methods.

CNN Optimization Discussion:

The FPGA operating parameters have been changed and optimized many times throughout the research made on CNN architecture. Additionally, there were changes involving quantization as well as fixed-point arithmetic so that while thinking about FPGA processing abilities, neural network learning ability is not compromised.

The existence of these outcomes demonstrates the achievement of this project in realizing its objectives and also giving direction for future hardware-based implementations of neural networks.

7.2 Future Directions and Recommendations:

The recent use of a resourceful field-programmable gate array (FPGA) in an improved Convolutional Neural Network (CNN), aimed at recognizing images, has promised to totally change hardware based visual data processing. This chapter will recommend additional steps that must be taken for the effective implementation and improvement of the system.

1. Enhancing the CNN Model:

Future work should improve upon the CNN model used by the system. More ultimate structures and diverse set of images for training purposes, as well as methods such as transfer learning are among other aspects that could be explored in refining this architecture. Better accuracy and efficiency of this model would make it more useful for real-time applications.

2. Optimization for FPGA Implementation:

Priorities in FPGA Implementation Optimization: The algorithm need to be further optimized for FPGA. This includes adjusting the model so that it can balance performance with resource constraints like limited memory and processing power found on FPGAs. Additional techniques to investigate include pruning, quantization, and efficient convolutional operations for models.

3. Integration of Additional Functionalities:

As a result, the system could be expanded to include image preprocessing in order to handle varying lighting conditions, orientations and scales. Moreover, directly incorporating FPGA based algorithms for enhancing images could greatly enhance the robustness and accuracy of the system.

4. Exploration of New Applications:

In addition, there is need for further research on face detection algorithms, object detectors as well as other non-visual data processing applications that may be integrated into FPGA-based systems, if it is desired to broaden its impact on the community. Thus every application might require specific modifications or optimizations of the current model.

5. Industry and Academia Collaboration:

Also, having such liaisons with technology companies and academic institutions can go a long way in terms of facilitating more advanced study as well as testing in real life situations. Besides these collaborations will help to ensure that this project remains up-to-date with respect to recent developments in FPGA technology and deep learning.

6. Long – Term Impact Studies:

Conducting long-term studies on the performance of this system in practice Situations may provide insight into its operational efficiency and durability. These studies can help to pinpoint Possible problems in continual use, and they can be used for increasing the stamina of such a system In summary, these are some future directions that will lead to improving FPGA-based CNN System for image recognition. Consequently, it could serve as

an effective solution for real-time image

Processing and analysis if it becomes more flexible and reliable by developing deeper learning models Continuously, upgrading their functionalities, experimenting with new applications & partners. Also, building This system will aid in addressing challenges and exploiting opportunities. Therefore, this research has the ability of exploring FPGA-based computing solutions' capabilities hence opening new frontiers in embedded systems with AI .

REFERENCES

[1]. Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015). **Optimizing FPGA-based accelerator design for deep convolutional neural networks**. Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

[2]. Suda, N., Chandra, V., Dasika, G., Mohanty, A., Ma, Y., Vrudhula, S., Seo, J. S., & Cao, Y. (2016). **Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks.** Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

[3]. Nurvitadhi, E., Venkatesh, G. M., Sim, J., Marr, D., Huang, R., Ong, Gee. A., & Liew, H. (2017). **Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?** Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

[4]. Qin, Y., Yu, Q., Li, L., & Zhang, W. (2018). **An FPGA-Based Parallelized Architecture for Deep Convolutional Neural Networks**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 26(12), 2781-2790. DOI: [10.1109/TVLSI.2018.2861807]

[5]. Motamedi, M., Fowers, J., Liu, G., & Weisz, G. (2016). **Design space exploration of FPGA-based deep convolutional neural networks.** Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 62-71. DOI: [10.1145/2847263.2847264]

[6]. Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., & Vissers, K. (2017). FINN: **A Framework for Fast, Scalable Binarized Neural Network Inference.** Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 65-74. DOI: [10.1145/3020078.3021744]

[7]. Aydonat, U., Ovtcharov, K., Fowers, J., Massengill, T., & Chou, P. A. (2017). **An OpenCL™ Deep Learning Accelerator on Arria 10**. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 55-64. DOI: [10.1145/3020078.3021741]

[8]. Venieris, S. I., & Bouganis, C. S. (2016). fpgaConvNet: **A Toolflow for Mapping Diverse Convolutional Neural Networks on FPGAs**. Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 217-226. DOI: [10.1145/2847263.2847279]

[9]. Guan, Y., & Zhang, C. (2018). **A Survey of FPGA-Based Accelerators for Convolutional Neural Networks.** Journal of Computer Science and Technology, 33(5), 768-792. DOI: [10.1007/s11390-018-1864-8]

[10]. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). **Gradient-based learning applied to document recognition**. Proceedings of the IEEE, 86(11), 2278-2324. DOI: [10.1109/5.726791]

[11]. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). **ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems**, 25, 1097-1105. DOI: [10.1145/3065386]

[12]. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., & Chen, T. (2018). **Recent advances in convolutional neural networks. Pattern Recognition**, 77, 354-377. DOI: [10.1016/j.patcog.2017.10.013]

APPENDEX A

SUSTAINABLE DEVELOPMENT GOALS FOR FYP

FYP Title :

FPGA Implementation of Optimized Deep Learning Algorithm

FYP Supervisor: Dr. Usman Ali_

Group Members:

Sr.No	Registration Number	Name
1	246935	Abdullah Benyamin
2	332352	Muhammad Kashif
3	346173	Nauman Saeed
4	348656	Zoraiz Ahmad

SDG's:

Sr.No	SDG No.	Justification After Consulting
1	4	Quality Education: Facilitating Access to learning opportunities
2	9	Industry Innovation: Developing Technology using FPGA contributes to industrial innovation.
3		
4		
5		

FYP Advisor Signature:_____

APPENDEX B

Final Year Project as Complex Engineering Problem:

Knowledge Profiles:

WK3: A systematic theory-based formulation of engineering fundamentals required in the engineering discipline. Our project involves understanding and applying fundamental principles of digital design, signal processing, and deep learning.

WK5: Knowledge that supports engineering design in a practice area. Your project involves designing a CNN and implementing it on an FPGA, which requires specific engineering design knowledge.

WK6: Knowledge of engineering practice (technology) in the practice areas in the engineering discipline. Implementing a deep learning algorithm on FPGA requires practical knowledge of both hardware and software aspects of engineering.

WK8: Engagement with selected knowledge in the research literature of the discipline. Your project likely involves reviewing and applying recent research findings in deep learning and FPGA technology.

Work Process:

WP1: Depth of Knowledge Required. The project requires in-depth knowledge in multiple areas including deep learning, digital design, and hardware-software co-design.

WP2: Range of conflicting requirements. The project involves managing conflicting requirements such as maximizing performance while minimizing resource usage and power consumption.

WP3: Depth of analysis required. The project requires abstract thinking and originality in both algorithm optimization and hardware implementation to formulate suitable models.

WP7: Interdependence

Your project involves high-level problems with many interdependent components, such as algorithm optimization, hardware design, and resource management. Changes in one area, like improving CNN accuracy, can affect FPGA resource use and power consumption. Achieving efficiency requires a holistic approach, considering all these interdependencies.

	WK1	WK2	WK3	WK4	WK5	WK6	WK7	WK8
WP1			X		X	X		X
WP2			X		X	X		X
WP3			X		X	X		X
WP4								
WP5								
WP6								
WP7			X		X	X	X	X

APPENDEX C

Plagiarism Report

Abth

ORIGINALITY REPORT

11%	8%	5%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	3%
2	link.springer.com Internet Source	1%
3	www.etsmtl.ca Internet Source	<1%
4	Ephrem Wu, Xiaoqian Zhang, David Berman, Inkeun Cho, John Thendean. "Compute-Efficient Neural-Network Acceleration", Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '19, 2019 Publication	<1%
5	arxiv.org Internet Source	<1%
6	Ezer, Erdem. "Kolonoskopi Görüntülerindeki Poliplerin Evrişimli Sinir Ağları Ile Tespiti", Marmara Universitesi (Turkey), 2023 Publication	<1%

7	dr.ntu.edu.sg Internet Source	<1 %
8	export.arxiv.org Internet Source	<1 %
9	Submitted to University of Prince Mugrin Student Paper	<1 %
10	Xin He, Jiawen Liu, Zhen Xie, Hao Chen, Guoyang Chen, Weifeng Zhang, Dong Li. "Enabling energy-efficient DNN training on hybrid GPU-FPGA accelerators", Proceedings of the ACM International Conference on Supercomputing, 2021 Publication	<1 %
11	escholarship.org Internet Source	<1 %
12	www.semanticscholar.org Internet Source	<1 %
13	Alberto Rivas, Pablo Chamoso, Alfonso González-Briones, Juan Corchado. "Detection of Cattle Using Drones and Convolutional Neural Networks", Sensors, 2018 Publication	<1 %
14	scholarbank.nus.edu.sg Internet Source	<1 %
15	Submitted to Glyndwr University Student Paper	<1 %

16	Submitted to University of Nottingham Student Paper	<1 %
17	nemertes.library.upatras.gr Internet Source	<1 %
18	qspace.library.queensu.ca Internet Source	<1 %
19	pdfs.semanticscholar.org Internet Source	<1 %
20	rosap.ntl.bts.gov Internet Source	<1 %
21	www.etonline-digitallibrary.com Internet Source	<1 %
22	Submitted to Arab Open University Student Paper	<1 %
23	cihr.ca Internet Source	<1 %
24	files.elektroda.pl Internet Source	<1 %
25	ir.busitema.ac.ug Internet Source	<1 %
26	www.springerprofessional.de Internet Source	<1 %
27	M. Joseph. "Floating-Point Adder in Techology Driven High-Level Synthesis",	<1 %

	Communications in Computer and Information Science, 2011 Publication	
28	cajmtcs.centralasianstudies.org Internet Source	<1 %
29	suspace.su.edu.bd Internet Source	<1 %
30	www.bgu.ac.il Internet Source	<1 %
31	www.ijariit.com Internet Source	<1 %
32	www.iris.unina.it Internet Source	<1 %
33	www.scholink.org Internet Source	<1 %
34	vdoc.pub Internet Source	<1 %
35	www.itbusinessedge.com Internet Source	<1 %
36	www.locus.ufv.br Internet Source	<1 %
37	Aymen Jemaa, Ons Zarrad, Mohamed Ali Hajjaji, Mohamed Nejib Mansouri. "Hardware Implementation of a Fuzzy Logic Controller for a Hybrid Wind-Solar System in an Isolated	<1 %

	Site", International Journal of Photoenergy, 2018 Publication	
38	S. Srilakshmi, G.L. Madhumati. "A Comparative Analysis of HDL and HLS for Developing CNN Accelerators", 2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS), 2023 Publication	<1 %
39	api.repository.cam.ac.uk Internet Source	<1 %
40	dspace.aiub.edu Internet Source	<1 %
41	github.com Internet Source	<1 %
42	mdpi-res.com Internet Source	<1 %
43	spiral.imperial.ac.uk Internet Source	<1 %
44	umpir.ump.edu.my Internet Source	<1 %
45	www.coursehero.com Internet Source	<1 %
46	www.mdpi.com Internet Source	<1 %

47	Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory", Neural Computation, 1997 Publication	<1 %
48	"Machine Learning and Big Data Analytics Paradigms: Analysis, Applications and Challenges", Springer Science and Business Media LLC, 2021 Publication	<1 %
49	"Proceedings of the Second International Conference on Advances in Computing Research (ACR'24)", Springer Science and Business Media LLC, 2024 Publication	<1 %
50	Pawar, Suraj. "Physics-Guided Machine Learning for Turbulence Closure and Reduced-Order Modeling", Oklahoma State University, 2023 Publication	<1 %

****