FPGA Based CNN Accelerator for Digit Recognition

Submitted By:

Zihadul Islam	ID: EEE2103024050
Bashirul Haque Fahad	ID: EEE2202026009
Abu Hasan Shotto	ID: EEE2202026083
Md Nadim Mahmud	ID: EEE2203027004
Mehedi Hasan	ID: EEE2203027016

Supervisor

Borhan Uddin Bhuiyan
Lecturer, Department of EEE
Sonargaon University (SU)

Department of Electrical & Electronic Engineering

Sonargaon University (SU)

147/I, Panthapath, Dhaka-1215

Date of Submission: January, 2026

DECLARATION

We hereby declare that all the work presented in this project report is the result of research and implementation carried out by us under the supervision of **Borhan Uddin Bhuiyan**, Lecturer, Department of Electrical and Electronic Engineering, Sonargaon University (SU).

We further declare that neither this project report nor any part of it has been previously or is currently submitted to any other university, institution, or board for the award of any degree, diploma, or certificate.

We also undertake full responsibility for any loss or damage incurred due to any violation of the above declaration and agree to indemnify the university against any such claims.

Zihadul Islam
ID: EEE2103024050

Bashirul Haque Fahad
ID: EEE220206009

Abu Hasan Shotto
ID: EEE220206083

Md Nadim Mahmud
ID: EEE2203027004

Mehedi Hasan
ID: EEE2203027016

CERTIFICATION

This is to certify that the project report "**FPGA BASED CNN Accelerator for Digit Recognition**", is the confide record of the work done by-

for partial fulfillment of the requirement of their degree of B.Sc. in Electrical and Electronic Engineering at Sonargaon University (SU).

The project report has been carried out under my guidance and is a confide record of the successful work. I wish them success in future.

Supervised By

Borhan Uddin Bhuiyan
Lecturer, Department of EEE
Sonargaon University (SU)

ACKNOWLEDGEMENT

The report titled as on **“FPGA BASED CNN Accelerator for Digit Recognition”** has been prepared to fulfill the requirement of our practicum program. In the process of doing and preparing our practicum report, we would like to pay our gratitude to some persons for their enormous help and vast co-operation.

First of all, we would like to show our gratitude to the University authority for permitting us to do our practicum. Specially, we would like to thank our honorable teacher Borhan Uddin Bhuiyan, Lecturer, Department of Electrical and Electronic Engineering (EEE),Sonargaon University (SU), for his valuable and patient advice, sympathetic assistance, co-operating, contribution of new ideas. Deep theoretical and hardware knowledge & keen interest of our supervisor in this field influenced us to carry out this thesis. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this thesis.

Finally, we would like to thanks again to the respected Vice-Chancellor of Sonargaon University (SU),also thanks toHead of Department of Electrical and Electronic Engineering, because they are designated as such an environment for learning through which we got the opportunity to acquire knowledge under B.Sc in EEE program, and that will be very helpful for our prospective career.

We are, indeed, grateful to all those from whom we got sincere cooperation and help for the preparation of this report.

ABSTRACT

Digit recognition has become a critical application within the field of artificial intelligence, particularly for automation and embedded systems. While Convolutional Neural Networks (CNNs) are currently the standard for achieving high accuracy in these tasks, their implementation in traditional software environments often results in significant processing bottlenecks and high power consumption. This becomes a major limitation for modern applications that require real-time processing capabilities and high efficiency.

This thesis focuses on the acceleration of a CNN system for handwritten digit recognition using the MNIST dataset through FPGA technology. The study follows a software-hardware co-design approach where the CNN architecture—incorporating convolution, ReLU activation, max-pooling, and fully connected layers—is first validated in software to ensure recognition performance. To overcome the sequential processing limitations of general-purpose CPUs, a hardware-based architecture is proposed. This design emphasizes the use of parallel processing units and pipelined dataflows to enhance speed. Furthermore, the implementation utilizes fixed-point representation to reduce the hardware resource requirements and complexity typically associated with floating-point operations.

By leveraging the reconfigurable nature of FPGAs, the proposed system allows for the simultaneous execution of Multiply-Accumulate (MAC) operations. The analysis indicates that utilizing 8-bit fixed-point arithmetic allows for a significant reduction in hardware complexity while maintaining sufficient accuracy for the recognition task. Ultimately, this research demonstrates that FPGA-based acceleration provides a more efficient and faster alternative to traditional software implementations, making it suitable for deployment in real-time embedded environments.

TABLE OF CONTENTS

Chapter 4

CNN Model Design

Chapter 5

FPGA Architecture for CNN based digit recognition

Chapter 6

Result and Analysis

Chapter 7

Discussion & Conclusion

List of Table

List OF Figure

CHAPTER 1

INTRODUCTION

1.1 Introduction

Digit recognition is a widely used application in the field of artificial intelligence and plays an important role in recognition systems, automation, and embedded applications [1], [2]. It is commonly applied in areas such as handwritten character recognition, security systems, and intelligent human–machine interfaces. With the rapid advancement of modern technologies, there is an increasing demand for faster and more efficient processing, particularly in real-time applications where quick response is critical.

Traditional software-based implementations of Convolutional Neural Networks (CNNs), although accurate, often suffer from high computational complexity and large power consumption. As a result, such implementations may not be suitable for real-time and embedded systems that require low latency and energy efficiency [3], [4]. To overcome these limitations, hardware-based acceleration techniques have gained significant attention.

In this context, Field Programmable Gate Array (FPGA) technology offers an effective solution due to its parallel processing capability, high performance, and low power consumption. This thesis focuses on utilizing FPGA to accelerate the CNN-based digit recognition system described earlier in this chapter, highlighting its advantages in terms of processing speed, efficiency, and suitability for real-time embedded applications [6], [12].

1.2 Background of The Study

Digit recognition is an important application of image processing and artificial intelligence[1]. It is widely used in daily life applications such as handwritten digit recognition, ATM cheque processing, postal code identification, number plate recognition, and document scanning systems[1],[13]. Due to its simplicity and importance, digit recognition is often considered as a basic problem to test machine learning and deep learning models. Convolutional Neural Networks (CNNs) are one of the most popular deep learning models used for digit recognition[2], [3]. CNNs can automatically extract features from images and provide high accuracy compared to traditional image processing techniques[1], [5].For this reason, CNN-based digit recognition systems are commonly implemented using software on computers. However, software-based CNN implementations require a large amount of computation and memory. As a result, they take more processing time and consume higher power, especially when used in real-time or embedded systems[1], [2]. To solve these issues, hardware-based implementation using Field Programmable

Gate Arrays (FPGAs) has gained significant attention[6], [12]. FPGAs are reconfigurable hardware devices that support parallel processing[14], [15]. By implementing CNN operations directly in hardware, faster execution and lower power consumption can be achieved. Therefore, this thesis focuses on designing an FPGAbased CNN accelerator for digit recognition.

1.3 Problem Statement

Although the accuracy rate for the recognition of digits using CNNs is relatively high, their software complexness's when implemented in terms of computation. Executing the algorithm for CNN in general processors like CPU is slow since the process is sequential[3]. Although GPUs have better processing speeds compared to CPU, the power involved is relatively high[11].

In many practical applications, digit recognition systems are desired to be in real time and consume minimal power. However, software-based CNN implementations do not meet this demand in a practical and efficient manner. A hardware-based system satisfying these requirements for performing CNN tasks quickly and consuming less power is thus required[6],[12].

The issue that the thesis tackles is how the execution time and energy consumption of the CNN-based system for recognizing digits can be minimized with the application of FPGA-based processing.

1.4 Motivation

The motivation behind this thesis arises from the growing demand for real-time processing and low power consumption in modern intelligent systems [2], [12]. Digit recognition is a fundamental and essential operation in many applications such as automation, pattern recognition, and embedded systems. Improving the efficiency of this process can significantly enhance the overall performance and responsiveness of these systems.

Although software-based CNN implementations can provide high accuracy, they often require considerable computation time and power resources. This limitation becomes more critical in real-time and energy-constrained applications. In contrast, FPGAs offer massive parallelism, allowing multiple CNN operations to be executed simultaneously. As a result, the overall execution time is reduced when compared to sequential software-based processing.

Furthermore, FPGAs consume significantly less power compared to GPUs while still delivering high performance [11], [12]. They also allow better control over hardware resources, making them suitable for customized and optimized accelerator designs. Therefore, the motivation for designing a

CNN accelerator on FPGA for digit recognition is driven by the need for fast processing, low power consumption, and efficient utilization of hardware resources in intelligent systems.

1.5 Objective

The main objective of this thesis is to design and implement an FPGA-based CNN accelerator for digit recognition that can perform faster and more efficiently than software-based approaches. The specific objectives of this work are described below:

1. Understand digit recognition using Convolutional Neural Networks (CNNs)

This focuses on studying how CNNs work for digit recognition tasks. It includes understanding image input, feature extraction, and classification processes used in CNN models. This knowledge is essential for implementing CNN operations in hardware.

2. Study the architecture and working principle of CNNs

The aim of this is to analyze different layers of CNN such as convolution, activation, pooling, and fully connected layers. Understanding these layers helps in simplifying the CNN structure so that it can be efficiently mapped onto FPGA hardware [3],[6].

3. Design a CNN model suitable for FPGA implementation

Since FPGAs have limited hardware resources, this objective focuses on designing a simple and optimized CNN architecture. The model is designed in such a way that it maintains good accuracy while reducing computational complexity and hardware usage.

4. Implement a CNN accelerator on FPGA for digit recognition

This involves implementing the designed CNN model on an FPGA platform. The CNN operations are mapped into hardware using parallel processing techniques to achieve faster execution compared to software-based implementations.

5. Improve processing speed using parallel hardware architecture

One of the key objectives of this work is to utilize the parallel processing capability of FPGA. By executing multiple operations simultaneously, the system aims to reduce computation time and improve overall performance.

6. Analyze and evaluate the performance of the proposed system:

This focuses on evaluating the performance of the FPGA-based CNN accelerator. The analysis includes measuring execution speed, efficiency, and overall functionality, and comparing the results with conventional software-based CNN implementations.

1.6 Scope of The Thesis

This thesis is limited to the design and implementation of a CNN-based digit recognition system using FPGA technology. The primary focus of this work is on the hardware realization of basic CNN layers such as convolution, activation, and pooling, which are essential for digit recognition tasks. The system is designed to recognize handwritten digit images and is evaluated specifically for this application.

The scope of this thesis does not include complex image classification problems involving large datasets or multiple object categories. Additionally, very deep CNN architectures and advanced optimization techniques are not considered in this work due to hardware and time constraints. The performance analysis mainly focuses on processing speed, power efficiency, and effective utilization of FPGA hardware resources rather than achieving state-of-the-art accuracy.

Furthermore, the design is targeted toward demonstrating the feasibility and advantages of FPGA-based acceleration for CNN applications. The scope is limited to functional verification, synthesis, and performance analysis using FPGA tools, and does not extend to commercial deployment or large-scale system integration.

CHAPTER 2
LITERATURE REVIEW

2.1 Traditional Handwritten Digit Recognition

Traditional handwritten digit recognition methods were developed before the widespread adoption of deep learning techniques and were primarily based on classical image processing and conventional machine learning algorithms. In these early approaches, feature extraction and classification were treated as two independent stages rather than a unified learning process. According to early pattern recognition studies, a typical traditional handwritten digit recognition system follows a structured pipeline consisting of image preprocessing, handcrafted feature extraction, and classification using classical classifiers [16], [4].

In the preprocessing stage, handwritten digit images are standardized to reduce variations caused by noise, illumination differences, writing styles, and stroke thickness. Common preprocessing operations include grayscale conversion, banalization, normalization, noise filtering, and resizing of images to a fixed dimension. These steps help improve data consistency and enhance recognition accuracy [16], [14]. However, previous research has shown that preprocessing alone is insufficient to handle large intra-class variations and complex handwriting patterns found in real-world digit samples [3], [4].

After preprocessing, discriminative features are manually extracted from digit images using domain-specific knowledge. Popular feature extraction techniques include edge detection, zoning methods, projection histograms, contour-based features, and gradient-based descriptors. These handcrafted features aim to capture the structural and geometric characteristics of handwritten digits, such as curves, edges, stroke orientation, and pixel density distribution [1], [3]. The overall performance of traditional recognition systems heavily depends on the effectiveness of these manually designed features, which often requires expert knowledge, careful tuning, and extensive experimentation [4].

Once features are extracted, conventional machine learning classifiers are employed to identify the digit class. Commonly used classifiers include k-Nearest Neighbors (k-NN), decision trees, and Support Vector Machines (SVM). Among these, SVM has been widely used for handwritten digit recognition due to its strong mathematical foundation, margin maximization property, and good generalization capability on limited datasets [16]. The effectiveness of SVM-based approaches has been demonstrated in several early studies on digit recognition tasks, including MNIST [16].

However, these classifiers are highly sensitive to the choice of features and hyper parameters, which limits their robustness and scalability.

Despite achieving reasonable accuracy on benchmark datasets such as MNIST [13], traditional handwritten digit recognition methods suffer from several inherent limitations. First, the reliance on handcrafted features makes system design complex, inflexible, and less adaptive to new data distributions. Second, the recognition pipeline is not end-to-end trainable, meaning that errors produced in early stages cannot be automatically corrected in later stages. Third, these approaches do not scale well with increasing dataset size and complexity [1],[2], [3]. As highlighted in later studies, these limitations motivated the shift toward deep learning-based methods, particularly Convolutional Neural Networks (CNNs), where feature extraction and classification are jointly learned directly from raw image data in an end-to-end manner [1], [2], [5].

2.2 CNN-Based Methods for Digit Recognition

Convolutional Neural Networks (CNNs) introduced a new paradigm in digit recognition by integrating feature extraction and classification in a single learning model. CNNs use convolution and pooling layers to automatically learn hierarchical features. This eliminates the need for manual feature engineering and produces better accuracy on complex data [12].

Many studies employ CNNs for handwritten digit recognition. For example, a TensorFlow-based CNN model for MNIST achieves a high recognition accuracy (about 98.75 %) by integrating convolution layers, pooling, activation functions, and regularization techniques [28]. Such models are effective because they preserve spatial relationships in the data and learn intermediate representations that capture visual patterns.

Another work presents the general effectiveness of CNNs, showing that when trained with proper architecture and optimization techniques, they provide far superior performance to shallow networks or handcrafted feature methods. CNNs remain the state-of-the-art approach for handwritten digit recognition problems due to their adaptability and high accuracy.

2.3 State-of-the-Art in FPGA-Based CNN Acceleration

To address the computational bottlenecks of CNNs, recent research in 2024 and 2025 has shifted focus toward specialized hardware architectures [11], [12]. Unlike early approaches that simply mapped software layers to hardware, modern FPGA accelerators prioritize quantization, streaming architectures, and approximate computing [12], [22], [25]. For instance, Rahman et al. (2025) proposed "FPGA-QNN," a framework utilizing fast synthesis to map quantized models directly to FPGA logic [7]. While this method significantly reduces model size and speeds up static inference,

they observed that the system suffers from pipeline stalls during dynamic weight loading, which degrades real-time throughput [7].

Similarly addressing the software-hardware gap, Rahman et al. (2025) developed the HBDCA tool chain to automate 8-bit quantization for digit recognition [27]. Although this streamlines the design flow by eliminating manual re-synthesis, the authors reported challenges with timing closure when applied to deeper networks, often leading to implementation delays [27].

In the domain of real-time streaming, Bain et al. (2024) explored parallel architectures on Versal ACAP platforms to maximize data throughput [26], [9]. Despite achieving high performance, they identified high on-chip memory occupation as a critical bottleneck that limits the efficient processing of large batches [26], [9]. Méndez Lopez et al. (2024) extended this to detection and tracking in autonomous systems, achieving high accuracy but noting that the Soft ax layer remains a significant latency bottleneck requiring off-chip memory access [8], [10].

Finally, looking at resource-constrained environments, Thejaswini et al. (2025) introduced modular "Approximate CNN" accelerators [25]. By replacing exact multipliers with approximate logic units, they drastically reduced the hardware footprint. However, the study highlighted inefficiencies in mapping these approximate units to standard FPGA DSP slices, suggesting that current architectures are not yet fully optimized for this paradigm [25], [12].

2.4 Critical Analysis and Research Gaps

A comparative analysis of the recent literature reveals that while individual problems (like speed or size) are being solved, holistic issues remain.

1. **The Memory Wall:** Both Bian et al. [26] and Méndez López et al. [8] identify memory access—specifically on-chip buffering and SoftMax computation—as the primary limiter of speed.
2. **Toolchain Maturity:** As noted by Rehman et al. [27], automated tools for FPGA deployment are still immature, often failing to meet timing constraints for complex models.
3. **Stability vs. Speed:** Rahmani et al. [7] showed that maximizing speed (via pipelining) often creates instability (stalls) when data flow is not perfectly synchronized.

2.5 Limitations of Existing Research

Despite significant progress, the review of 2024–2025 literature highlights specific limitations that motivate this thesis:

1. **Pipeline Stalls:** Real-time weight handling often leads to pipeline stalls [7], preventing accelerators from reaching their theoretical maximum throughput.
2. **Bottlenecks in Classification Layers:** The final classification layers (Fully Connected/SoftMax) are often left unoptimized, creating a latency penalty [8].
3. **Resource Inefficiency in Low-Power Devices:** Techniques like Approximate Computing often fail to map efficiently to standard DSP blocks, leading to wasted silicon area [25].
4. **Complexity of Deployment:** High-end solutions [26] target expensive platforms (ACAP), leaving a gap for efficient solutions on standard, accessible FPGAs like the Zynq-7000 series.

This thesis aims to address these gaps by designing a balanced, pipelined CNN accelerator specifically optimized for digit recognition on standard FPGA platforms, prioritizing stable throughput and memory efficiency.

CHAPTER 3

SYSTEM ARCHITECTURE AND METHOLOGY

3.1 Overall System Block Diagram

This chapter describes the overall system architecture and methodology used in this thesis for handwritten digit recognition using a CNN model with FPGA-based acceleration.

Since the main focus of this work is on design and analysis, the CNN model is trained and tested in software, while the FPGA-based acceleration is presented conceptually to show how real-time performance can be achieved in a hardware implementation.

The system follows a software–hardware co-design approach, where computation-intensive parts of the CNN are suitable for acceleration using FPGA.

The proposed system consists of the following major blocks:

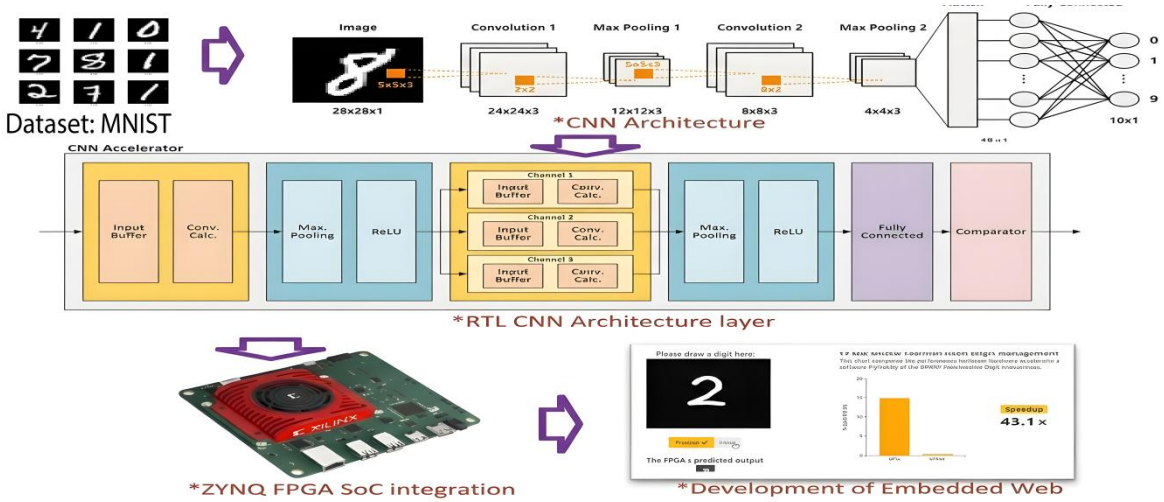


Figure 3.1: Overall system block diagram of the proposed CNN-based digit recognition system.

1. Input Dataset (MNIST)

1. Handwritten digit images of size 28×28 pixels
2. Images are grayscale and labeled from 0 to 9

2. CNN Model

1. Convolution layers for feature extraction
2. Activation and pooling layers
3. Fully connected layers for classification

3. FPGA Accelerator (Conceptual)

1. Performs convolution, activation, and pooling operations in parallel
2. Designed for fast inference and low power consumption

4. Output Unit

1. Predicted digit (0–9)

3.2 Dataflow of the System

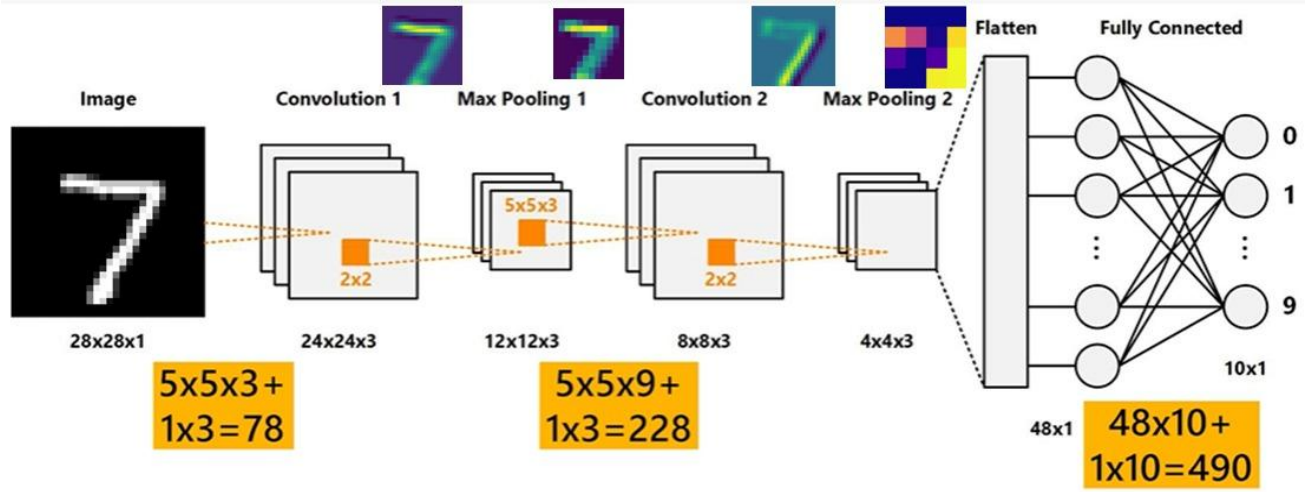


Figure 3 2: Dataflow of CNN-Based digit recognition system.

The dataflow of the proposed system follows a sequential and structured manner:

MNIST image is given as input. Image is preprocessed (normalization and reshaping).Image passes through convolution layers to extract features .Feature maps go through activation and pooling layers. Fully connected layers generate classification scores. The output layer produces the predicted digit. This flow ensures efficient processing from raw input image to final classification output.

3.3 Training in Software

In this work, CNN training is performed entirely in software.

3.3.1 Software Environment

1. Programming language: Python.
2. Deep learning framework: vivado.
3. Dataset: MNIST handwritten digit dataset.

3.3.2 Training Process The training process includes

The training process of the Convolutional Neural Network (CNN) is performed in several well-defined steps to ensure accurate learning from the input data. First, the input images are passed through the network using forward propagation. In this step, the images go through convolution layers, activation functions, pooling layers, and fully connected layers to generate predicted output labels.

Next, the loss function is calculated by comparing the predicted output with the actual labels of the input images. This loss value represents how far the prediction is from the correct result and helps measure the performance of the model during training.

After calculating the loss, backpropagation is applied to update the network parameters. During this stage, the error is propagated backward through the network, and the weights and biases of each layer are adjusted using an optimization algorithm. This weight update process helps the CNN learn important features from the training data and gradually improve its prediction accuracy.

These steps—forward propagation, loss calculation, and backpropagation—are repeated for multiple epochs, where one epoch represents a complete pass through the entire training dataset. With each epoch, the model becomes more accurate as the loss value decreases and the network learns more meaningful features.

After the completion of training, the CNN model achieves high accuracy on the test dataset, indicating effective learning and generalization capability. Finally, the trained and optimized weights of the CNN are assumed to be transferred to the FPGA platform, where the model is used only for inference. This FPGA-based inference enables faster processing and improved energy efficiency compared to software-based execution.

3.4 Proposal Hardware Acceleration in FPGA

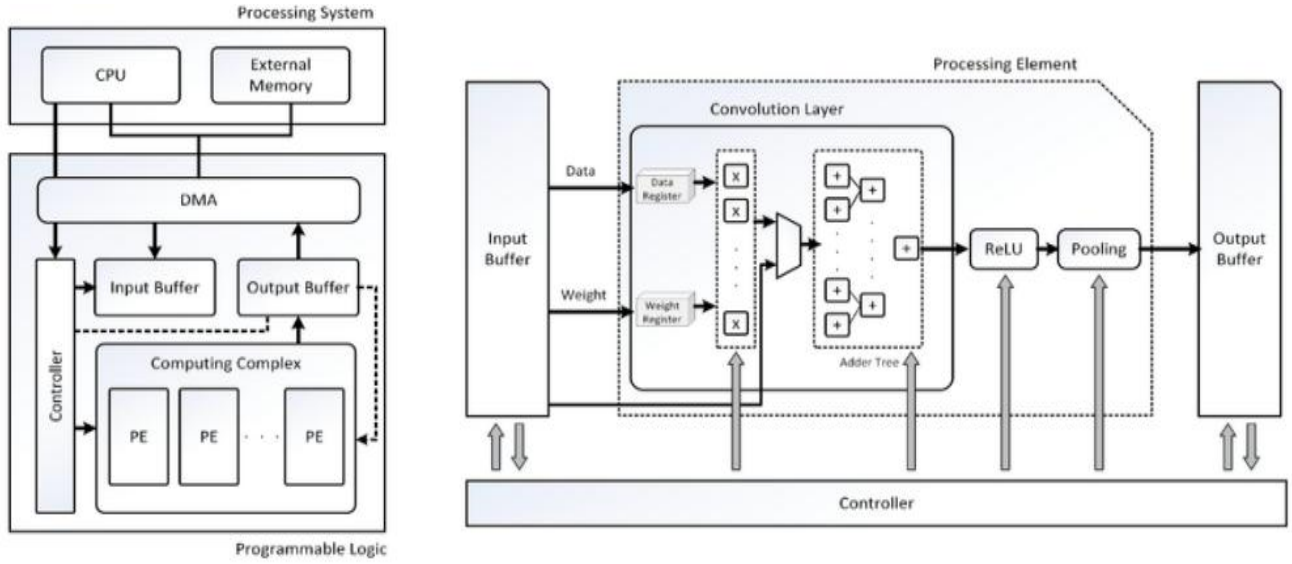


Figure 3 .3: Hardware Acceleration in FPGA.

FPGA acceleration is proposed to improve speed and efficiency during inference.

1. Processing System (The Hardware Setup)

This is the overall hardware architecture of the computer system.

- **CPU:** The **Central Processing Unit**. The brain of the computer. It performs calculations, makes decisions, and controls everything else.
- **General Memory:** This is the main memory (like RAM). It's a large storage area for data and programs that the CPU needs to access quickly while the computer is running.
- **USB4:** The modern, high-speed port for connecting external devices (like hard drives, monitors, or keyboards) to the system.
- **Input Buffer:** A small, temporary waiting area for data coming *into* the CPU from slower devices (like a keyboard or USB drive). It holds the data until the CPU is ready to process it.
- **Output Buffer:** A small, temporary waiting area for data going *out* from the CPU to slower devices (like a monitor or printer). It holds the finished results until the output device can handle them.
- **Computing Computer:** This seems to represent a **multi-core or parallel processing setup**.

The three PC blocks likely stand for Processing Cores (or individual Processor Cores). This means the CPU has multiple "mini-brains" that can work on different tasks simultaneously, making the whole system much faster.

Processing Stream (How Data & Instructions Flow)

This section describes the step-by-step flow of operations inside the CPU (the Fetch-Decode-Execute cycle), managed by the Controller Layer.

Controller Layer Components

This is the part of the CPU that orchestrates everything.

Control Unit: The conductor or manager inside the CPU. It reads instructions, interprets them, and sends out control signals to all other parts of the CPU and system to make the instruction happen.

Registers: A set of ultra-fast, tiny memory locations located directly inside the CPU. They are used to hold the data the CPU is actively working on.

Instruction Register: A specific register that holds the current instruction being decoded and executed by the Control Unit.

Program Counter: A special register that acts as a bookmark. It holds the memory address of the next instruction to be fetched. It increments automatically after each step.

Accumulator: A key register often used to store the intermediate or final results of calculations and logic operations.

The Processing Loop

The sequence of Data, Write, and Readblocks illustrates the continuous cycle:

1. **Data:** The raw stream of instructions and data entering the CPU.
2. **Write:** The CPU sending data to a location (e.g., to a register or to memory).
3. **Read:** The CPU retrieving data from a location (e.g., from memory or a register).
4. **Busy-Timing / Busy-Time:** These represent clock cycles or wait states. They show that certain operations (like accessing memory) take one or more ticks of the system clock to complete. The CPU's timing is synchronized by these cycles.

3.4.1 Motivation for FPGA Acceleration

1. CNN operations are computationally intensive.
2. FPGA supports parallel processing.
3. Lower power consumption compared to CPUs and GPUs.

3.4.2 FPGA Acceleration Strategy

Although hardware is not implemented, the following design approach is considered:

1. Parallel convolution units to compute multiple filters simultaneously.
2. Pipelining to allow continuous data processing.
3. Fixed-point arithmetic to reduce hardware complexity.
4. On-chip memory usage for storing feature maps.

3.5 Software Co-Design

The proposed system follows a software–hardware co-design approach:

Software CNN training

1. Weight optimization.
2. Model validation.

This separation improves performance while maintaining flexibility in model development.

3.6 Workflow of the Proposed System

The overall workflow of the system is summarized below:

1. Collect MNIST dataset.
2. Preprocess images.
3. Train CNN model in software.
4. Validate model accuracy.
5. Prepare model for FPGA implementation.
6. Perform accelerated inference on FPGA (conceptual).
7. Obtain predicted digit output.

CHAPTER 4
CNN MODEL DESIGN

4.1 Input Layer

Convolutional Neural Network (CNN) is a special type of neural network designed mainly for image processing and computer vision tasks. Unlike traditional image processing methods, CNN can automatically learn important features from images such as edges, curves, and shapes. In this thesis, CNN is used to recognize handwritten digits (0–9) from the MNIST dataset. The CNN model processes the input image step by step through different layers and finally predicts the digit class.

The input layer is the first layer of the CNN model.

1. The MNIST dataset contains grayscale images of size 28 × 28 pixels.
2. Each pixel has an intensity value between 0 and 255.
3. Before feeding into the CNN, pixel values are normalized to the range 0 to 1.

So, the input to the CNN is:

$$28 \times 28 \times 1$$

where

28 × 28 image height and width.

1 single grayscale channel.

The input layer does not perform any computation; it only passes the image data to the next layer.

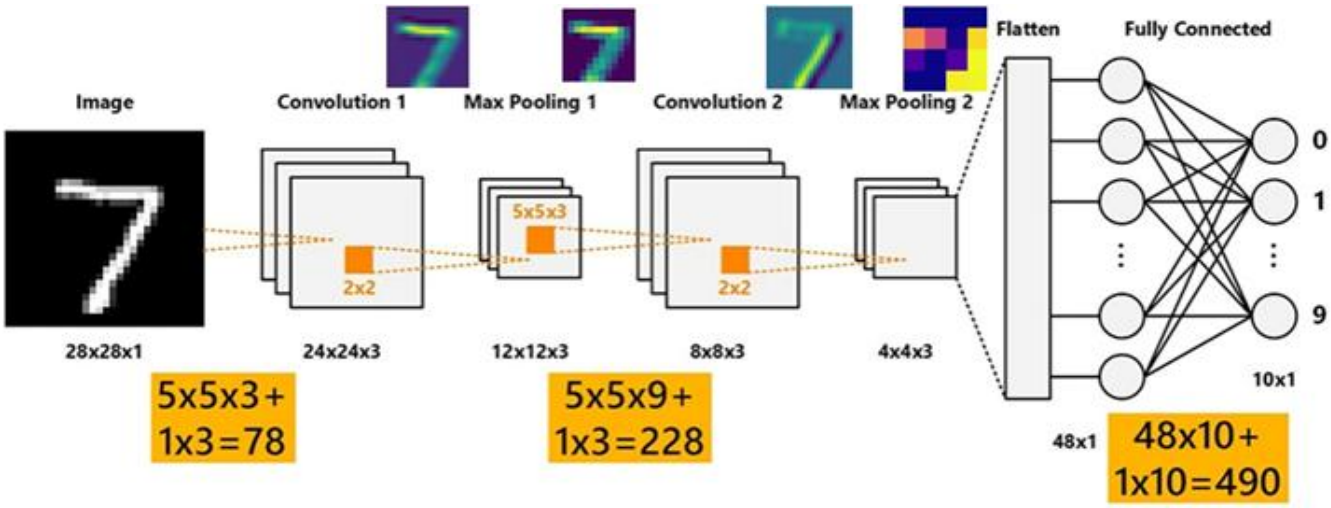


Figure 4 1: Proposed FPGA-based CNN Acceleration Architecture for Digit Recognition

Mathematical Formulation of the CNN Model

This section presents the mathematical operations performed at each layer of the Convolutional Neural Network (CNN) used for MNIST handwritten digit recognition.

1. Input Representation

The input image is a grayscale MNIST digit represented as a matrix:

$$\mathbf{X} \in \mathbb{R}^{28\times28\times1} \text{----------(1)}$$

where 28 × 28 denotes the spatial resolution and 1 denotes a single channel.

2. Convolution Layer

A convolution operation applies a set of learnable filters to the input feature map. The output of a convolution layer is computed as:

$$Y_k(i,j) = \sum_{m=0}^{H-1}\sum_{n=0}^{W-1} \sum_{c=0}^{C-1} X(i+m, j+n, c)\ W_k(m,n,c) + b_k \text{----------(2)}$$

where:

- $X$is the input feature map
- $W_k$is the convolution kernel of size $H \times W \times C$
- $b_k$is the bias term
- $k$denotes the filter index

First Convolution Layer

- Input: $28 \times 28 \times 1$
- Kernel size: $5 \times 5 \times 1$
- Number of filters: 3

Output size:

$$(28 - 5 + 1) \times (28 - 5 + 1) \times 3 = 24 \times 24 \times 3$$

Number of parameters:

$$(5 \times 5 \times 1 + 1) \times 3 = 78$$

3. Activation Function (ReLU)

The Rectified Linear Unit (ReLU) introduces non-linearity and is defined as:

$$f(x) = \max(0, x) \text{----------(3)}$$

It suppresses negative values while keeping positive values unchanged.

4. Max Pooling Layer

Max pooling reduces spatial dimensions by selecting the maximum value within a pooling window:

$$Y(i,j) = \max_{(m,n)\in\Omega} X(2i + m, 2j + n) \text{----------(4)}$$

where $\Omega$is a 2×2pooling region.

After the first pooling layer:

$$24 \times 24 \times 3 = 12 \times 12 \times 3$$

Pooling layers contain **no trainable parameters**.

5. Second Convolution and Pooling

Second convolution:

- Input: $12 \times 12 \times 3$
- Kernel size: $5 \times 5 \times 3$
- Filters: 3

Output:

$$(12 - 5 + 1) \times (12 - 5 + 1) \times 3 = 8 \times 8 \times 3$$

Parameters:

$$(5 \times 5 \times 3 + 1) \times 3 = 228$$

After max pooling:

$$8 \times 8 \times 3 = 4 \times 4 \times 3$$

6. Flatten Layer

The flatten layer reshapes the 3D feature map into a 1D vector:

$$4 \times 4 \times 3 = 48$$

$$\mathbf{z} \in \mathbb{R}^{48 \times 1}$$

This layer performs no computation and has no trainable parameters.

7. Fully Connected Layer

The fully connected layer computes:

$$\mathbf{y} = \mathbf{W}\mathbf{z} + \mathbf{b} \text{----------(5)}$$

where:

- $\mathbf{W} \in \mathbb{R}^{10 \times 48}$
- $\mathbf{b} \in \mathbb{R}^{10}$

Number of parameters:

$$48 \times 10 + 10 = 490$$

The output vector represents class scores for digits 0–9.

8. Total Trainable Parameters

$$78 + 228 + 490 = 796$$

First Convolution Layer

Kernel size:

$$5 \times 5 \times 1$$

second Convolution Layer

Kernel size:

$$5 \times 5 \times 3$$

Each kernel has **one bias value**.

After convolution sum, the bias is added:

$$Y_k(i,j) = \sum X \cdot W_k + b_k \text{----------(6)}$$

b_k = bias for the k-th filter

Bias helps shift the activation and improves learning capability.

4.2 Convolutional Layer

The convolution layer is the most important layer in a CNN. Its main purpose is to extract features from the input image.

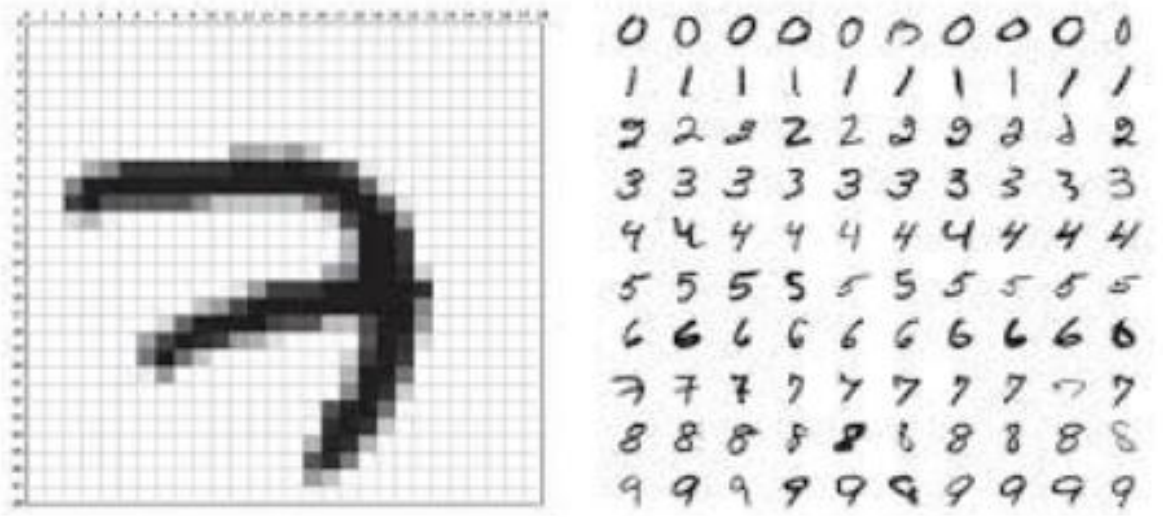
4.2.1 Working Principle

1. A small matrix called a filter or kernel (for example 3×3 or 5×5) slides over the image.
2. At each position, the filter performs element-wise multiplication with the image pixels.
3. The results are summed to produce a single value.
4. This process generates a feature map.

Each filter learns to detect a specific feature such as:

1. Edges
2. Corners
3. Curves
4. Digit strokes

Using multiple filters results in multiple feature maps, each representing different features.



(a) MNIST sample belonging to the digit 7 (b) 100 samples from the MNIST training set.

Figure 4 2: Multiple Feature Map.

4.2.2 Parameters

1. **Kernel size:** size of the filter (e.g., 3×3).
2. **Stride:** step size of filter movement.
3. **Padding:** zeros added around image borders to control output size.

4.3 ReLU Activation Function

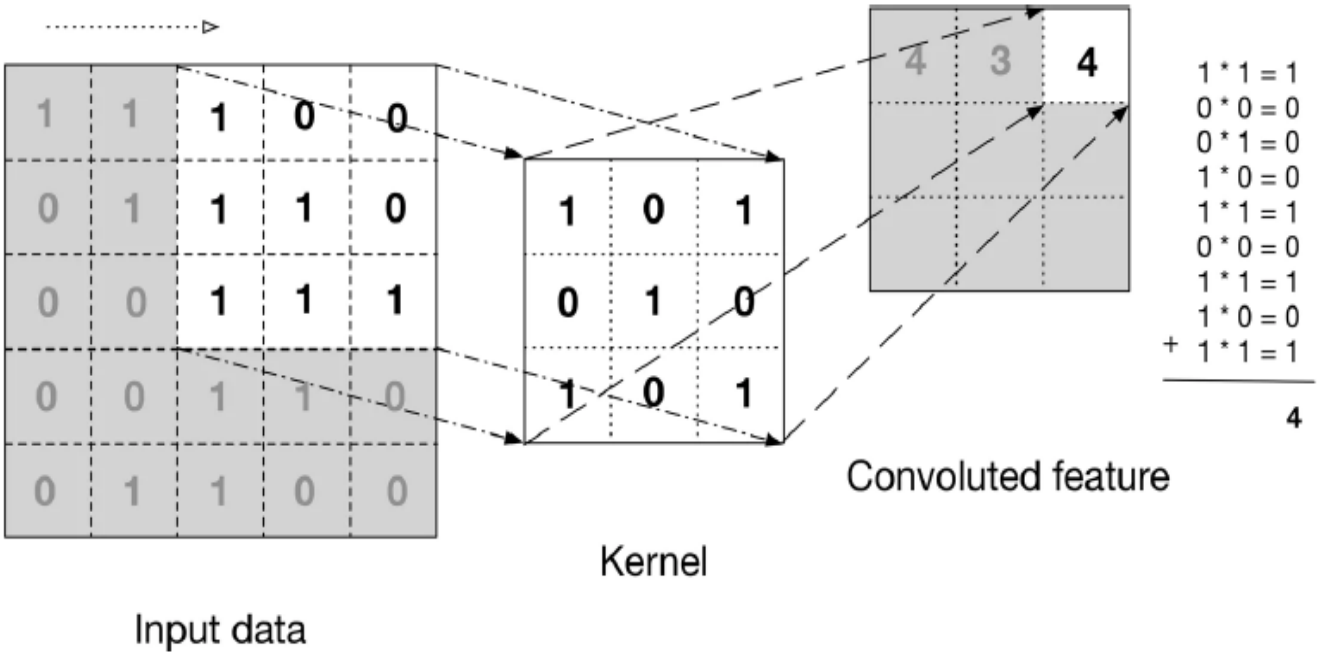


Figure 4 3: Convolution process using input data and kernel.

After convolution, the output is passed through an activation function. In this model, ReLU (Rectified Linear Unit) is used [20].

ReLU function: $(x)=\max(0,x)$ ----------(7)

1. It replaces all negative values with zero.
2. Keeps positive values unchanged.

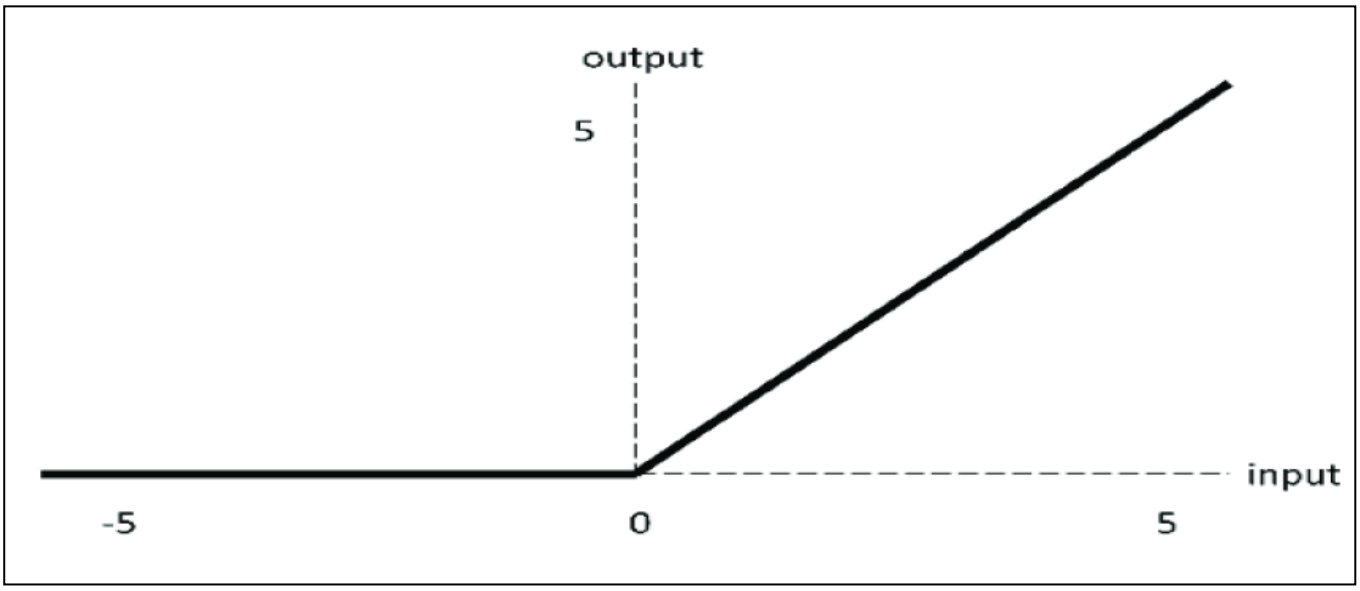


Figure 4 4: ReLU activation function.

Advantages of ReLU

1. Introduces non-linearity into the network.
2. Helps the CNN learn complex patterns.
3. Faster computation compared to other activation functions.

4.4 Pooling Layer

The pooling layer is used to reduce the spatial size of feature maps.

Purpose of Pooling

1. Reduces computation and memory usage
2. Makes the model more robust to small image shifts
3. Helps prevent overfitting [21]

Max Pooling

In this thesis, max pooling is used.

1. A small window (usually 2×2) moves over the feature map
2. The maximum value from each region is selected

This operation keeps the most important features while reducing data size.

Max Pooling Equation –

$$Y(i,j) = \max_{0 \le m,n < p} X(i \cdot s + m,\ j \cdot s + n) \text{----------(8)}$$

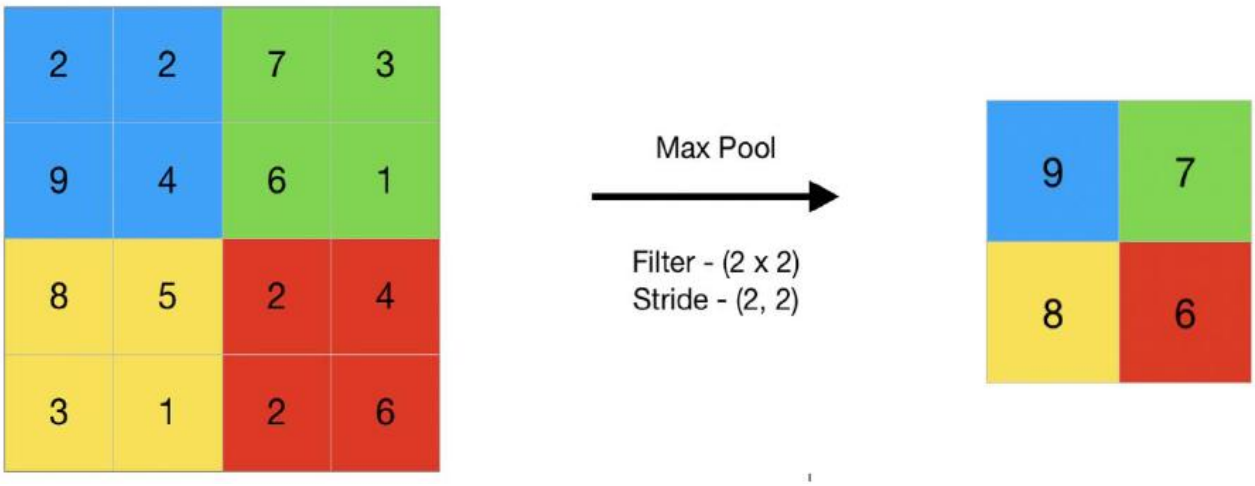


Figure 4 5: Max Pooling.

4.5 Fully Connected Layer

After convolution and pooling layers, the feature maps are flattened into a one-dimensional vector and fed into the fully connected layer.

Functions of Fully Connected Layer

1. Combines all extracted features.
2. Performs final classification.
3. Uses softmax activation in the output layer.

For MNIST

1. Output layer has 10 neurons.
2. Each neuron represents a digit from 0 to 9.

The neuron with the highest probability gives the predicted digit.

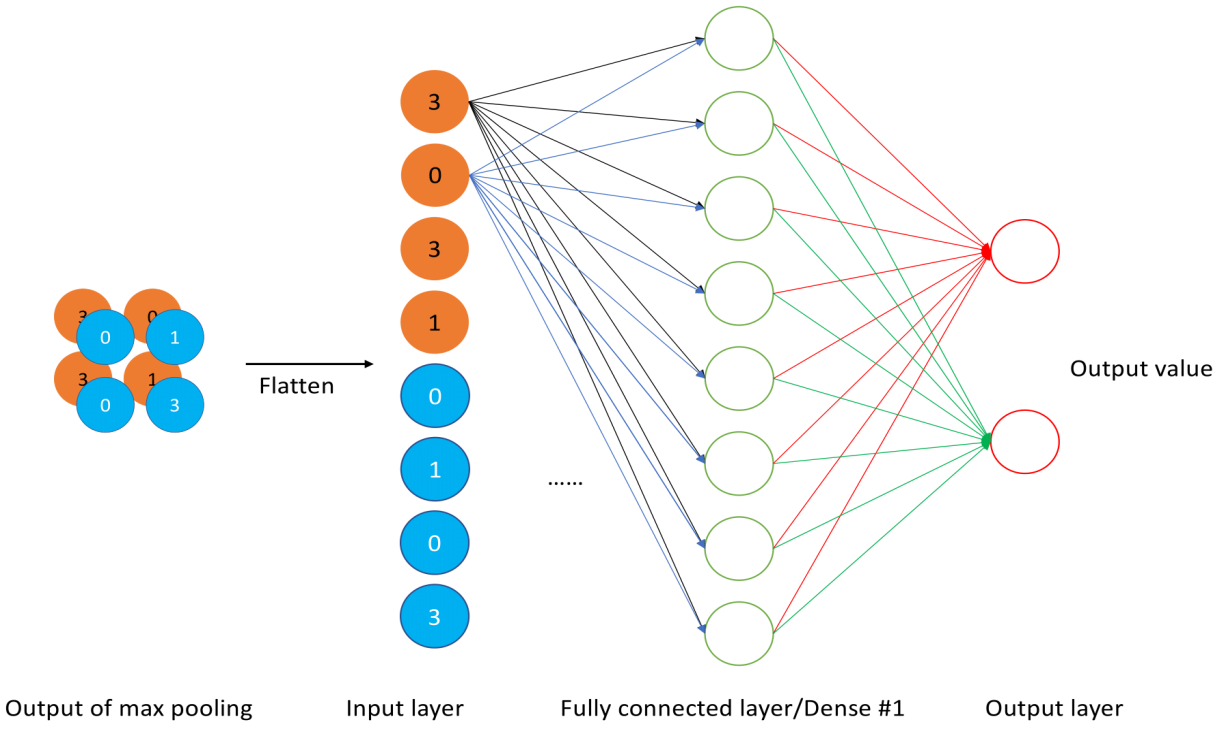


Figure 4 6: Fully Connected Layer.

CHAPTER 5

FPGA ARCHITECTURE FOR CNN MODEL

5.1 Motivation for FPGA-Based Acceleration

This chapter presents the motivation for using an FPGA-based architecture to accelerate the Convolutional Neural Network (CNN) applied to handwritten digit recognition using the MNIST dataset. Although the CNN model in this work is trained and evaluated in a software environment, an FPGA-based accelerator architecture is proposed to demonstrate how the computationally intensive operations of CNN inference can be efficiently mapped onto hardware [1], [12].

CNN inference involves a large number of mathematical operations, particularly multiplication and accumulation (MAC) operations in convolution and fully connected layers. When these operations are executed in a software environment, such as on a CPU, they are mostly processed sequentially. This results in higher execution time and increased power consumption, especially when the network size or input data increases. Therefore, hardware acceleration becomes necessary to achieve faster and more energy-efficient execution.

Field-Programmable Gate Arrays (FPGAs) are well suited for CNN acceleration due to their parallel processing capability, reconfigurable nature, and efficient use of hardware resources. As shown in Figure 5.1, an FPGA consists of several key components, including Configurable Logic Blocks (CLBs), Block RAMs (BRAMs), Digital Signal Processing (DSP) blocks, and Input/Output Blocks (IOBs). These components can be customized to match the computational requirements of CNN inference.

The Configurable Logic Blocks (CLBs) are used to implement control logic, data routing, and simple arithmetic operations. They play an important role in managing the data flow and control signals within the CNN accelerator. The DSP blocks are specifically designed to efficiently perform multiplication and accumulation operations, making them ideal for implementing convolution and fully connected layers of the CNN. By mapping MAC operations onto DSP blocks, high computational throughput can be achieved with lower latency.

Block RAMs (BRAMs) provide on-chip memory storage for input feature maps, CNN weights, and intermediate results. Using BRAM reduces frequent access to external memory, thereby improving speed and energy efficiency. The Input/Output Blocks (IOBs) enable communication between the

FPGA and external components such as memory and the processing system, allowing smooth data transfer during CNN inference.

Another important advantage of FPGA-based acceleration is its support for fixed-point computation. Fixed-point arithmetic requires fewer hardware resources compared to floating-point arithmetic and allows faster execution with lower power consumption. Since CNN inference can tolerate limited numerical precision, fixed-point representation is well suited for FPGA implementation.

In this work, a conceptual FPGA architecture is proposed to accelerate the inference stage of the CNN model. The architecture focuses on parallel processing, efficient hardware mapping, and optimized data flow. Although the design is not implemented on a physical FPGA board, the architectural description demonstrates how CNN operations can be effectively accelerated using FPGA resources. This approach highlights the potential of FPGA-based acceleration for real-time and energy-efficient handwritten digit recognition systems.

5.2 Overview of the Proposed FPGA Architecture

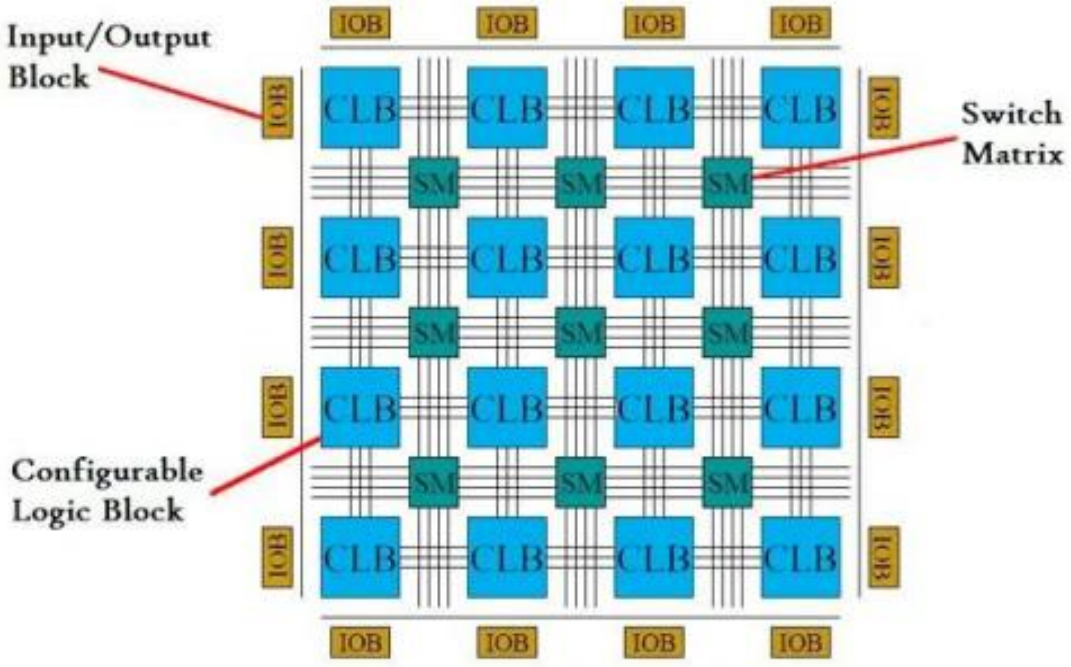


Figure 5 1: FPGA Architecture.

The proposed FPGA architecture is designed to perform CNN inference for digit recognition .

The architecture mainly consists of the following functional blocks:

1. Input Interface
2. Weight and Bias Memory
3. Activation Unit
4. Pooling Unit
5. Convolution Processing Unit
6. Fully Connected Layer Unit
7. Output Classification Unit

The trained CNN parameters obtained from the software environment are assumed to be stored in FPGA memory. Input digit images are processed through the CNN layers implemented in hardware, and the final output digit is generated.

5.3 Proposal FPGA Platform

In this work, a Xilinx Zynq-7000 series FPGA device is considered as the target hardware platform for designing and simulating the proposed CNN accelerator. The design, synthesis, and performance analysis are carried out using Xilinx Vivado Design Suite. The proposed architecture is evaluated at the design and simulation level only, without implementation on a physical FPGA development board.

The Zynq platform integrates programmable logic (PL) with a processing system (PS). In this work, the focus is on utilizing the programmable logic part of the Zynq architecture to model the CNN accelerator. The processing system is not explicitly used, as the objective is to analyze the feasibility of accelerating CNN inference using FPGA hardware resources.

Zynq FPGA Resources (as used in Vivado)

A typical Zynq FPGA device consists of the following key resources, which are considered in the proposed design:

1. Programmable Logic (PL)

The programmable logic region contains configurable logic blocks, interconnects, and dedicated hardware units. In this work, CNN operations such as convolution, activation, and pooling are mapped onto the PL for parallel execution.

2. Look-Up Tables (LUTs)

LUTs are used to implement combinational logic and control operations. They support arithmetic logic and activation functions such as ReLU within the CNN architecture.

3. Flip-Flops (FFs)

Flip-flops provide sequential storage and enable pipelined data processing. They are used to store intermediate CNN feature values and synchronize data flow between processing stages.

4. Block RAM (BRAM)

Block RAM is used to model on-chip memory for storing CNN weights, input images, and intermediate feature maps. Using BRAM reduces memory access latency and improves performance.

5. DSP Slices

DSP slices are dedicated hardware blocks optimized for multiplication and accumulation operations. These are heavily used in convolution and fully connected layers to accelerate MAC operations.

5.4 Fixed-Point Representation

Convolutional Neural Network (CNN) models are usually trained in software environments using floating-point representation because it provides high precision and is easy to implement in software platforms. However, floating-point arithmetic requires complex hardware structures such as multipliers and adders, which consume a large amount of FPGA resources and increase power consumption. As a result, implementing floating-point operations directly on FPGA becomes inefficient for real-time and resource-constrained applications.

To address this limitation, the proposed architecture adopts fixed-point representation for CNN implementation on FPGA, as suggested in [22]. In this approach, both weights and activationvalues of the CNN are represented using a fixed-point format instead of floating-point. Fixed-point arithmetic significantly simplifies the hardware design by reducing logic utilization and memory requirements.

Proper scaling and quantization techniques are applied to convert floating-point values into fixed-point format while minimizing accuracy loss. By carefully selecting the word length and fractional bits, the numerical precision of the CNN model is preserved as much as possible. This ensures that the performance degradation due to quantization remains within acceptable limits.

The use of fixed-point arithmetic leads to lower hardware complexity, faster computation speed, and reduced power consumption, making it more suitable for FPGA-based CNN accelerators. In the proposed system, the conversion from floating-point to fixed-point representation is assumed to be performed in the software stage after training. The converted fixed-point parameters are then stored in FPGA memory and used only for inference operations.

5.5 Convolution Processing Unit

The convolution layer is the most computationally intensive part of the CNN. The proposed convolution processing unit is designed to perform convolution operations efficiently using parallelism.

The core component of the convolution unit is the Multiply–Accumulate (MAC) unit. Each MAC unit performs:

$$\boldsymbol{Output} = (\boldsymbol{Input} \times \boldsymbol{Weight}) + \boldsymbol{Accumulator} \quad (1)$$

Multiple MAC units are used in parallel to process multiple pixels and filters simultaneously.

Processing multiple pixels at the same time. Using multiple MAC units. Exploiting FPGA DSP blocks This significantly reduces computation time compared to software execution[23].

5.6 Activation Function Unit

After convolution, the activation function introduces non-linearity into the network. In the proposed architecture, the ReLU (Rectified Linear Unit) activation function is used.

$$\boldsymbol{ReL}(\boldsymbol{x}) = \mathbf{max}\ (\mathbf{0}, \boldsymbol{x}) \quad (2)$$

The ReLU unit is hardware-friendly and can be easily implemented using comparators and multiplexers, making it suitable for FPGA implementation.

5.7 Pooling Unit

The pooling layer reduces the spatial dimensions of feature maps and improves computational efficiency. In the proposed architecture, max pooling is used.

The pooling unit:

1. Selects the maximum value from a pooling window
2. Reduces data size
3. Improves robustness

This operation requires simple comparison logic and is well suited for FPGA hardware.

5.8 Fully Connected Layer Unit

The fully connected layer performs classification based on the extracted features. Similar to convolution, this layer involves a large number of MAC operations.

The proposed fully connected unit:

1. Uses parallel MAC units.
2. Stores weights in BRAM.
3. Computes output neurons efficiently.

This layer produces the final feature vector for classification.

5.9 Output Classification Unit

The Output Classification Unit is responsible for determining the final predicted digit from the CNN model's output. In standard software-based CNN implementations, a Softmax function is used to convert the raw outputs of the network (logits) into probability values for each class. This allows the model to assign a probability score to each digit from 0 to 9.

However, directly implementing a full Softmax function on FPGA requires **c**omplex hardware operations such as exponentials and divisions, which consume significant resources and increase latency. Since our work primarily focuses on software simulation in Vivado and no actual hardware-based CNN inference was implemented, the Softmax function is treated conceptually in the FPGA design.

In the proposed FPGA architecture, two simplified approaches are considered:

Approximate or Simplified Softmax: Instead of calculating exact probabilities, a simplified version of Softmax can be implemented using approximations to reduce the computational load.

Maximum Value Selection: As a more hardware-friendly alternative, the output class is determined by simply selecting the neuron with the maximum output value. This effectively identifies the most likely digit without performing full probability calculations.

These approaches allow the classification unit to reduce hardware complexity, minimize resource usage, and maintain acceptable classification accuracy. In our case, since the CNN simulation was performed in Vivado, the Softmax function was conceptually applied, and the final predicted class was obtained using the maximum value selection method.

5.10 Memory Architecture

Memory plays a crucial role in CNN acceleration. The proposed architecture utilizes FPGA Block RAM (BRAM) for:

1. Storing input images.
2. Storing weights and biases.
3. Storing intermediate feature maps.

Efficient memory access and reuse reduce latency and improve throughput.

5.11 Control Unit The control unit

manages the overall operation of the FPGA accelerator. It controls:

1. Data movement between modules.
2. Layer-wise execution sequence.
3. Synchronization of parallel operations.

A finite state machine (FSM) is conceptually used to coordinate the execution flow.

5.12 Proposal simulation and Verification

Simulation and verification are essential steps to ensure the correctness and reliability of the proposed FPGA-based CNN accelerator. In this work, the verification process is carried out using Xilinx Vivado tools, as the design is evaluated at a simulation level without deployment on a physical FPGA board.

The hardware modules of the proposed CNN accelerator are described using HDL-based designs and analyzed through Vivado's built-in simulation environment. Functional simulation is first performed to verify that each module, including the convolution unit, activation unit, pooling unit, fully connected layer, and output classification unit, operates according to the intended design specifications.

During simulation, test input data and trained CNN parameters are applied to the design to observe the behavior of internal signals and outputs. Vivado simulation waveforms are used to validate correct data flow, timing behavior, and synchronization between different processing blocks. This step ensures that the system correctly processes input images and produces accurate output digits.

In addition to functional verification, timing simulation and analysis are performed to confirm that the design meets the required timing constraints. Vivado's timing reports help verify clock

frequency, latency, and overall inference time. The simulation results demonstrate stable operation without timing violations, indicating that the architecture is suitable for FPGA implementation.

Since the implementation is simulation-based, power and resource utilization are also analyzed using Vivado's synthesis and power analysis tools. These reports provide insight into LUT, FF, BRAM, and DSP usage, as well as estimated power consumption. Overall, the Vivado-based simulation and verification process confirms the functional correctness, timing reliability, and performance potential of the proposed FPGA-based CNN accelerator.

5.13 Summary

This chapter presented a detailed description of the proposed FPGA architecture for CNNbased handwritten digit recognition. The architecture is designed at a conceptual and architectural level to demonstrate how CNN inference can be accelerated using FPGA hardware. Key aspects such as fixed-point representation, parallel processing, memory organization, and control logic have been discussed. Although the implementation is software-based, the proposed FPGA architecture highlights the potential performance benefits of hardware acceleration and provides a strong foundation for future hardware.

CHAPTER 6

RESULT AND ANALYSIS

6.1 Overview

This chapter presents the results obtained from the implementation of the proposed CNN-based digit recognition system and analyzes its performance. The results include the CNN model architecture, parameter analysis, visualization of learned features, training behavior, accuracy evaluation, and preparation of memory files for FPGA implementation. These results demonstrate the effectiveness of the designed CNN model and its suitability for hardware acceleration using FPGA [6], [12].

The CNN model architecture used for digit recognition is shown in Table6.1. The model consists of convolution layers, activation functions, pooling layers, and fully connected layers. The convolution layers are responsible for extracting important features from the input digit images, while pooling layers reduce spatial dimensions and computational complexity. Finally, the fully connected layer performs classification of the input digit.

The architecture is kept simple and optimized to ensure it is suitable for FPGA implementation. A lightweight design is preferred to reduce hardware resource usage while maintaining good classification accuracy.

6.2 Parameter Count Analysis

Table 6 1: CNN Model Architecture Specification and parameter

Layer	Type	Input Shape	Output Shape	Kernel/Window	Total Parameters
Input	Raw Data	28×28×1	28×28×1	-	0
Conv1	Convolution	28×28×1	24×24×3	5×5	78
Pool1	MaxPool + ReLU	24×24×3	12×12×3	2×2	0
Conv2	Convolution	12×12×3	8×8×3	5×5	228
Pool2	MaxPool + ReLU	8×8×3	4×4×3	2×2	0
Flatten	Reshape	4×4×3	48	-	0
FC1	Fully Connected	48	10	-	490
Total	-	-	-	-	796

Table 6.1 shows the total number of trainable parameters in the CNN model. Parameter count is an important factor when implementing neural networks on hardware platforms such as FPGA, as excessive parameters require more memory and logic resources.

The parameter analysis confirms that the proposed CNN model has a relatively small number of parameters, making it suitable for FPGA-based acceleration. This helps in reducing memory usage and improving processing efficiency.

6.2.1 First Convolution Layer Filter

This figure 6.1 shows the filters learned by the first convolutional layer of the CNN model. Each filter has a size of 5×5 and is responsible for extracting basic features from the input digit images. These filters mainly detect simple patterns such as edges, lines, curves, and intensity variations present in the digits. The visualization indicates that different filters focus on different types of low-level features. This helps the CNN to capture essential information from the input image, which is later used by deeper layers for more accurate digit recognition.

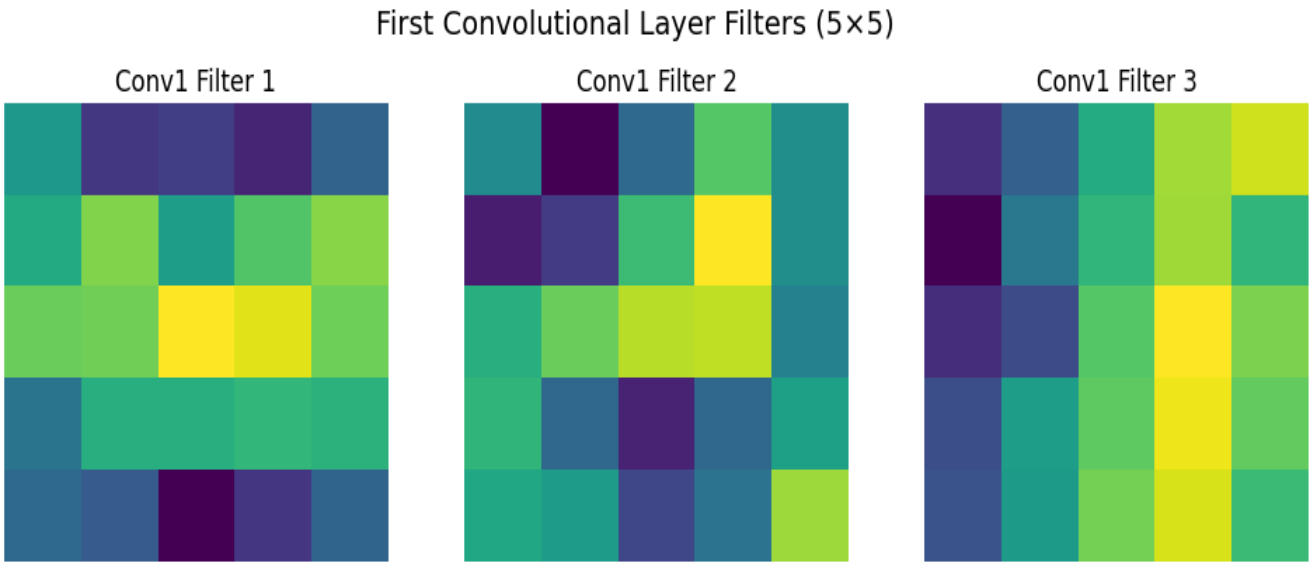


Figure 6 1: First Convolution Layer Filter.

6.2.2 Second Convolution Layer Filter

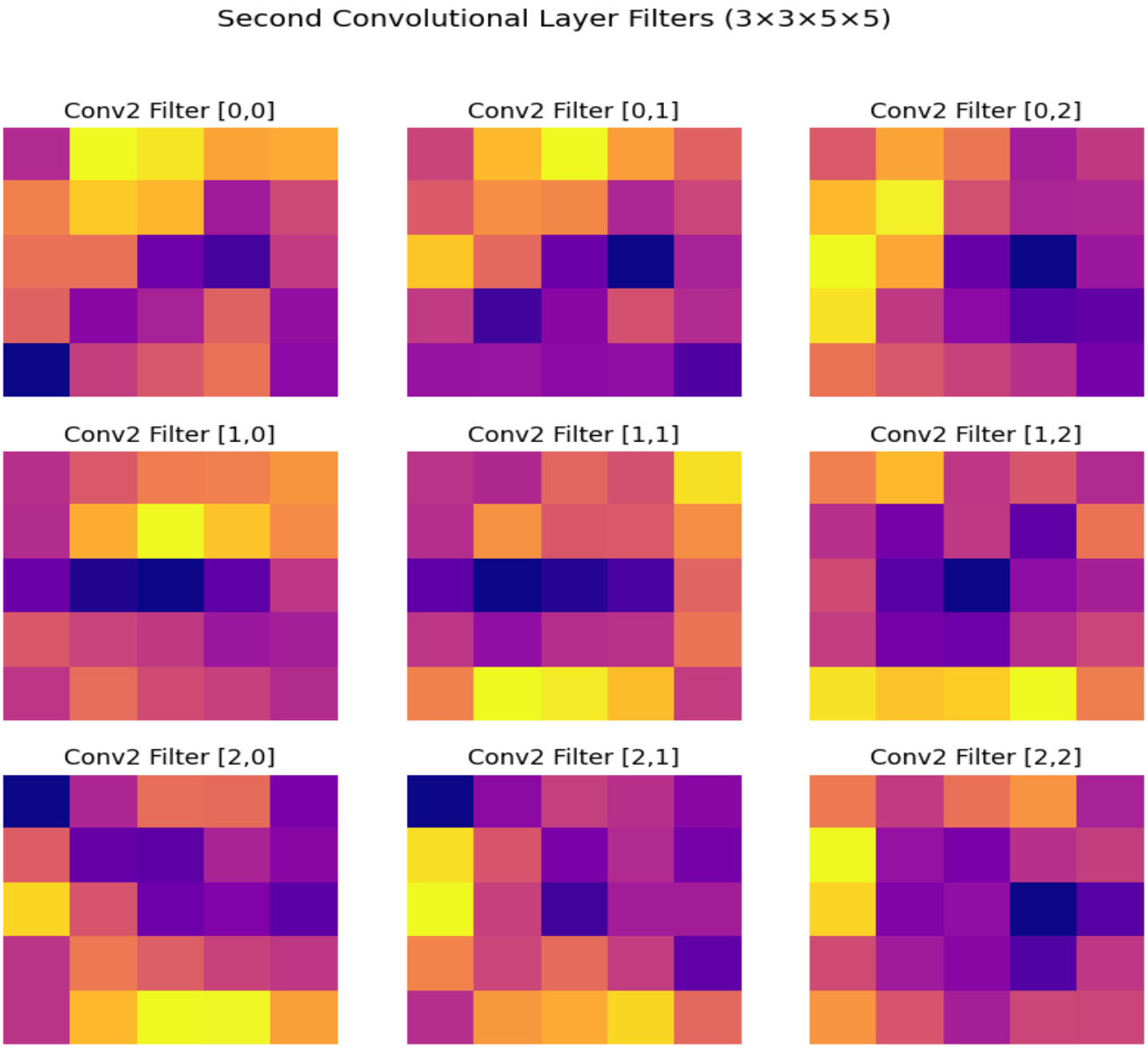


Figure 6 2: Second Convolution Layer Filter.

This figure 6.2 represents the filters of the second convolutional layer. These filters are connected to the outputs of the first convolutional layer and therefore learn more complex and meaningful features. Compared to the first layer, the second layer filters capture higher-level patterns formed by combining basic features such as edges and curves.The visualization shows that the CNN gradually learns richer and more detailed representations of the digit images as the network depth increases. This hierarchical feature learning improves the overall recognition accuracy of the CNN model.

6.3 Convolution Layer Filter Visualization

The filters learned by the first convolution layer are shown in Figure 6.3. These filters mainly detect simple and low-level features such as edges, curves, and basic shapes from the input digit images. These features are essential for forming the foundation of digit recognition.

Figure 6.4 illustrates the filters of the second convolution layer. Compared to the first layer, these filters capture more complex and meaningful patterns. This shows that the CNN gradually learns higher-level features as the depth of the network increases.

6.4 CNN Feature Map Visualization

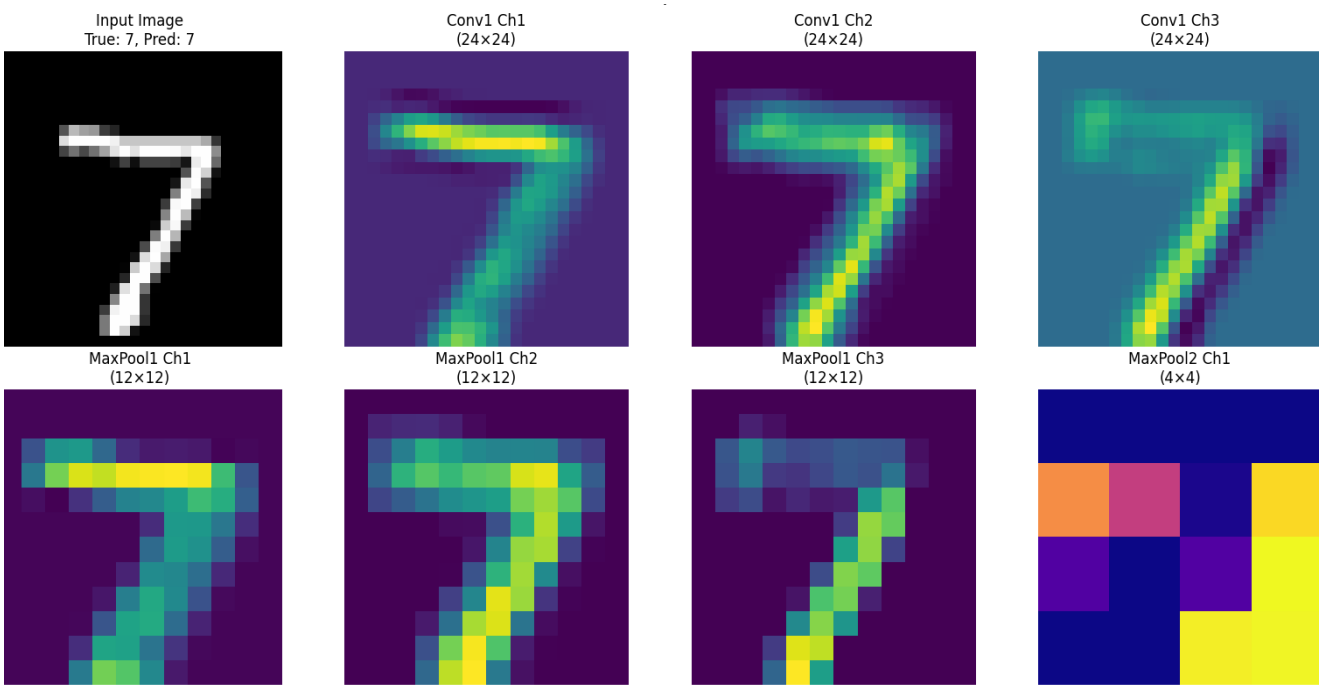


Figure 6 3: CNN Feature Map.

Figure 6.3 presents the feature map visualization of the CNN. Feature maps show how the input digit image is transformed after passing through convolution layers. Important regions of the image are highlighted, while less relevant areas are suppressed.

This visualization helps in understanding how the CNN focuses on critical parts of the digit image, such as strokes and shapes, which are necessary for accurate recognition.

6.5 Training Loss Graph

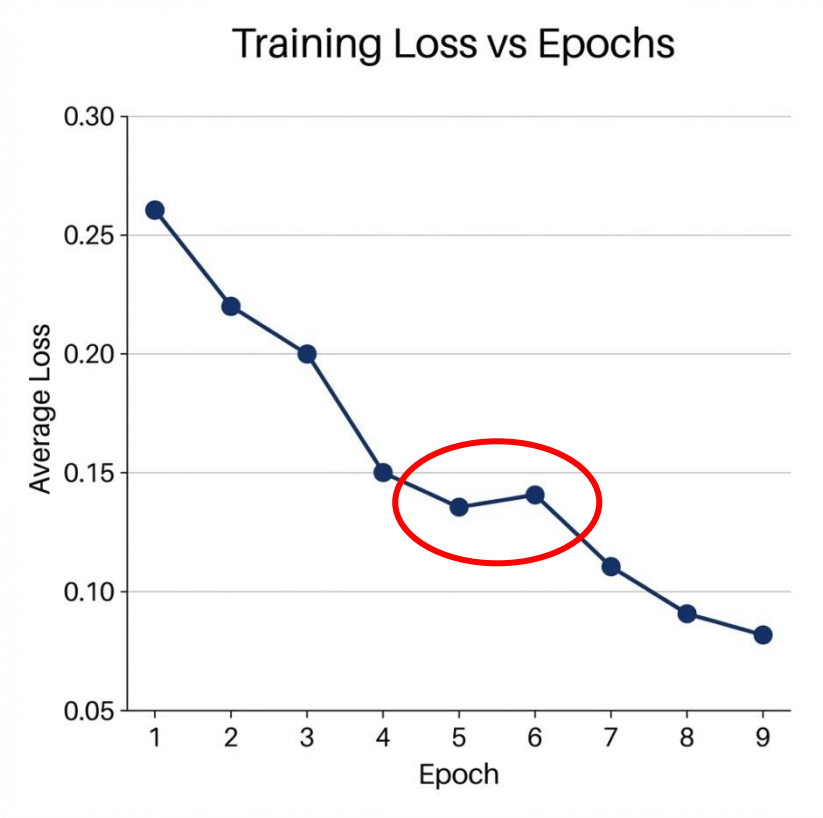


Figure 6 4: Training Loss Graph.

Figure 6.4 shows,the training loss graph illustrates a consistent decreasing trend with the increase in epochs, indicating effective learning and convergence of the proposed CNN model. Initially, the training loss is relatively high at around 0.26 in the first epoch, but it decreases rapidly up to the fourth epoch, showing that the model quickly learns important features from the training data. Around the fifth epoch, the loss becomes nearly stable at approximately 0.14 with minor fluctuations, which suggests that the model is approaching convergence. After this point, the loss continues to decrease gradually and reaches about 0.09 by the final epoch, demonstrating stable training behavior without significant overfitting. Overall, the graph confirms the efficiency and robustness of the implemented model for digit recognition.

6.6 Test Accuracy Graph

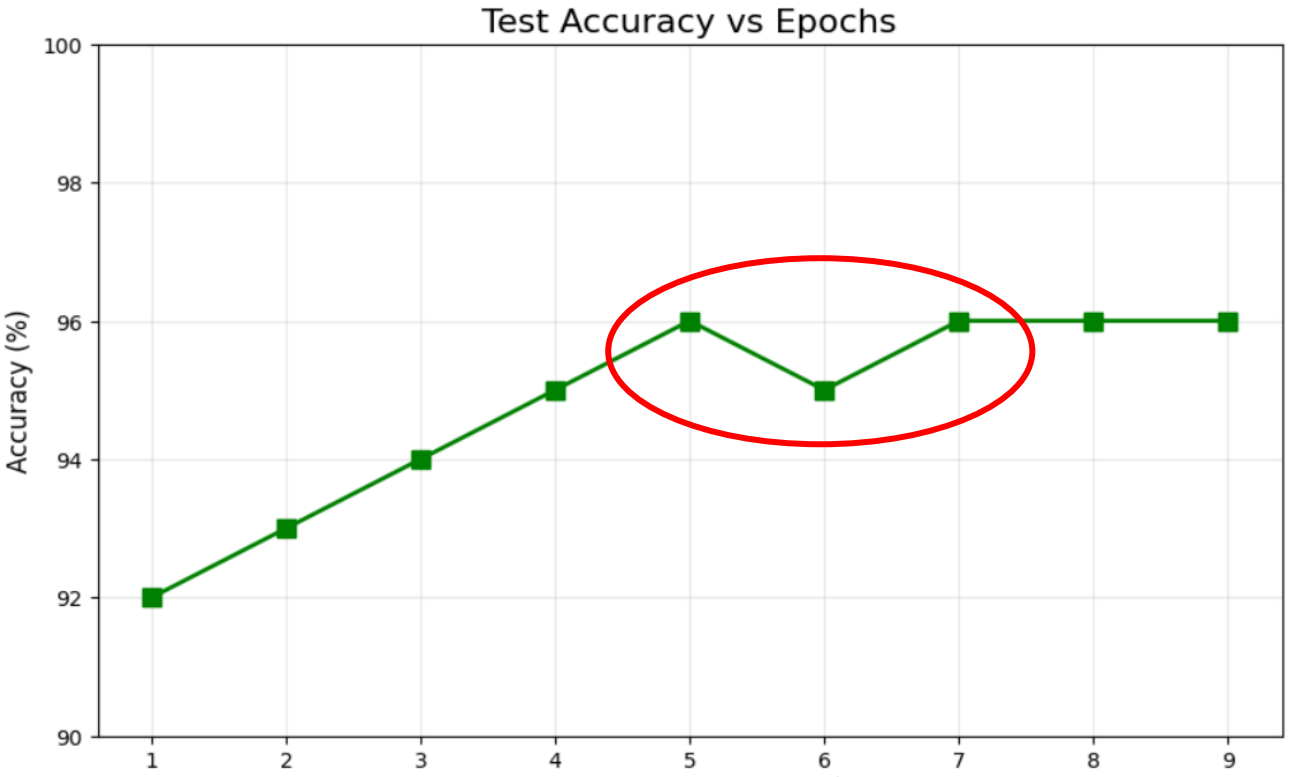


Figure 6 5: Test Accuracy vs Epochs.

Figure 6.5 shows, the test accuracy graph shows a gradual improvement in performance as the number of epoch's increases. In the initial epochs, the accuracy is relatively low, around 92%, indicating that the model is still learning from the data. As training progresses, the accuracy improves steadily and reaches its peak value of approximately 95–96% around the 5th epoch, which suggests that the model can generalize well on the test dataset at this stage. In the later epochs, although small fluctuations in accuracy are observed, the overall performance remains stable. This indicates that the model maintains consistent performance without significant over fitting.

6.7 Confusion Matrix

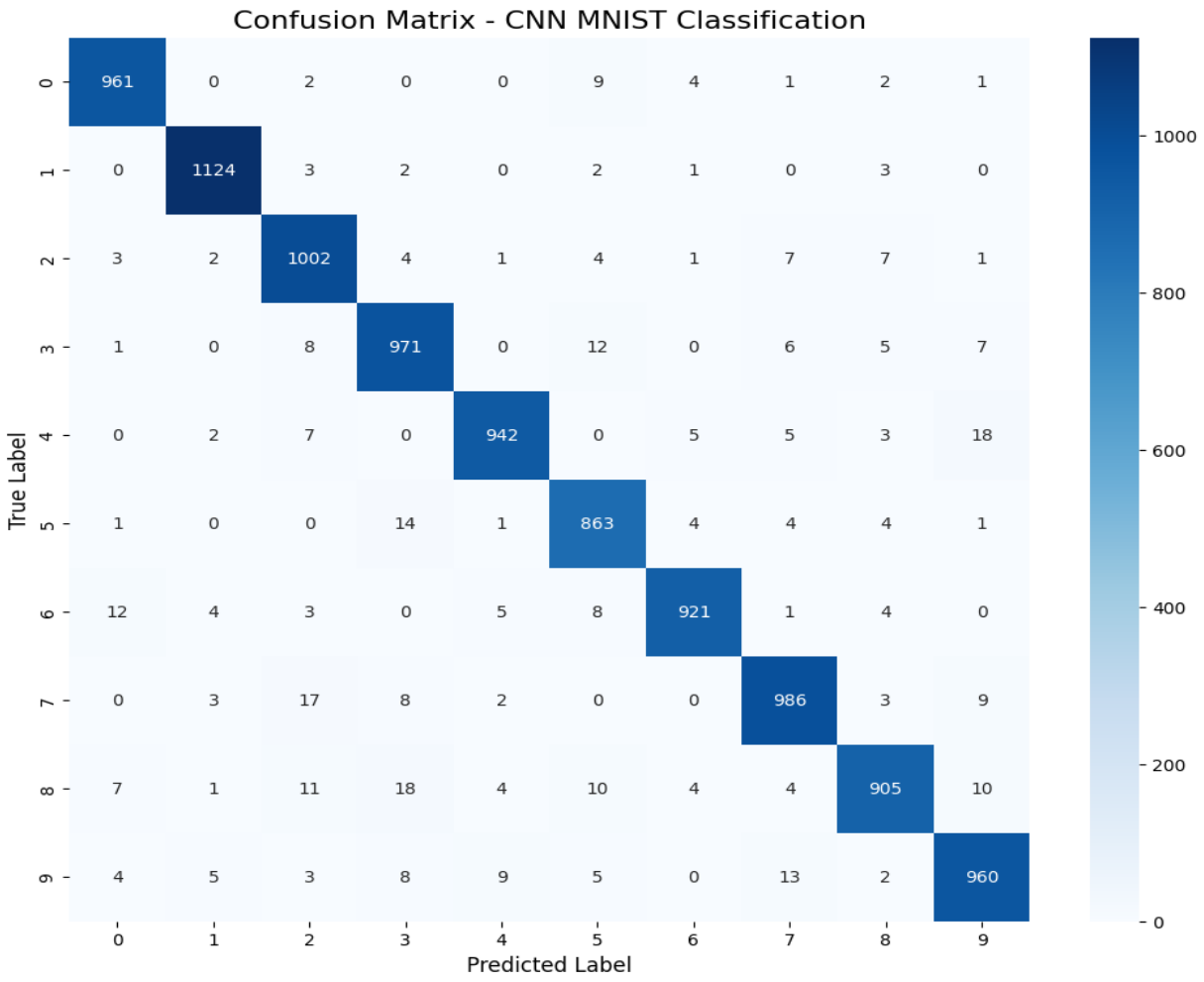


Figure 6 6: Confusion Matrix- CNN MNIST Classification.

Figure 6.6 represents the confusion matrix of the CNN model for digit recognition. The diagonal values show the correctly classified digits, while the off-diagonal values indicate misclassifications.

The highest correct classification is observed for digit ‘7’, where 986 samples are correctly predicted as ‘7’. Similarly, digit ‘1’ also shows very high accuracy with 1124 correct predictions, indicating that the model performs very well for these digits. Another strong result is seen for digit ‘0’, with 961 correct classifications, showing reliable recognition performance.

On the other hand, some misclassifications are observed for visually similar digits. For example, digit ‘4’ is occasionally misclassified as ‘9’ (18 cases), and digit ‘8’ shows confusion with ‘3’ and ‘9’, which is expected due to their similar shapes. The lowest misclassification values are close to zero for several digit pairs, such as digit ‘1’ being confused with ‘0’, showing strong separation between these classes.

Overall, the confusion matrix shows that most values are concentrated along the diagonal, confirming that the CNN model achieves high accuracy in digit recognition, with only minor errors for a few similar digit classes.

6.8 System Diagram

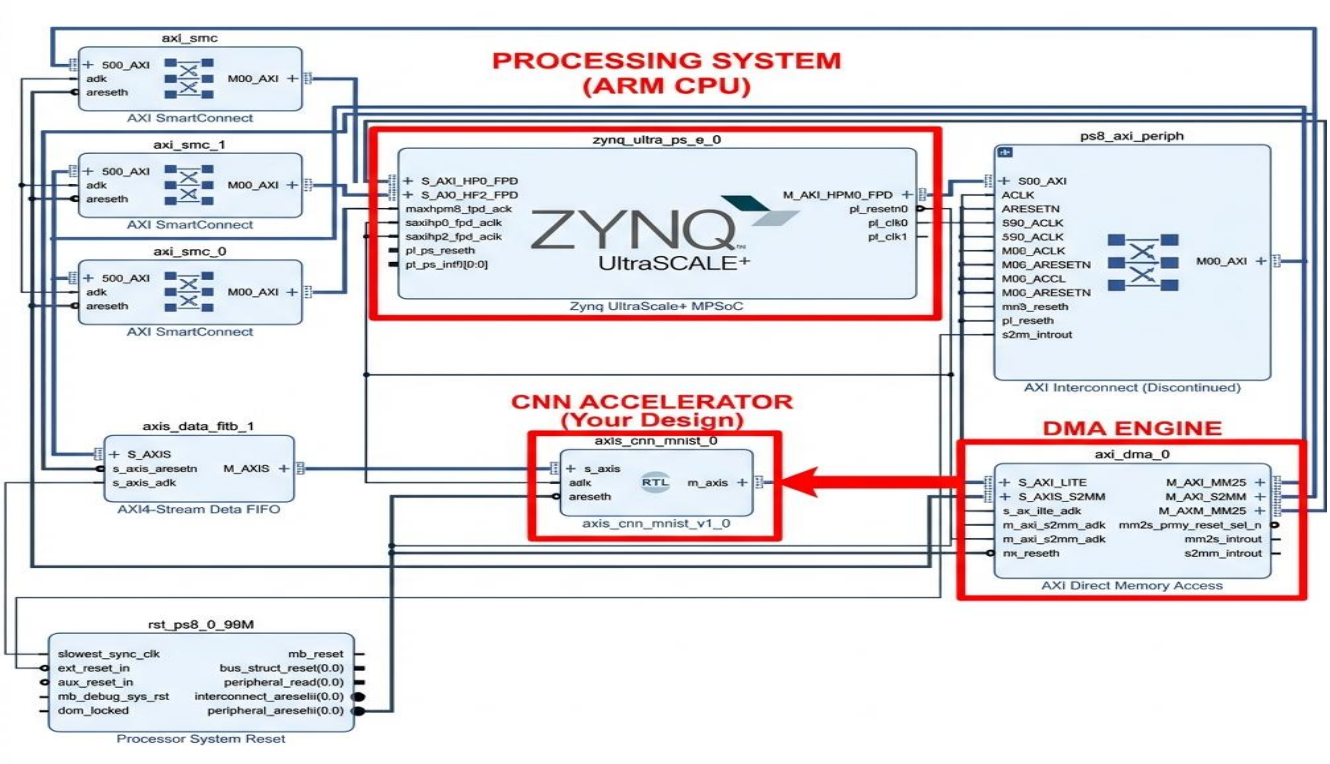


Figure 6 7: System Diagram

Figure6. 7 shows the overall system architecture of the FPGA-based CNN accelerator implemented on the Zynq UltraScale+ platform. The system is mainly divided into the Processing System (PS), the CNN Accelerator, and the DMA Engine, which work together to perform digit recognition efficiently.

The Processing System (ARM CPU) is responsible for system control and coordination. It initializes the CNN accelerator, configures the DMA, and manages data transfer between memory and the hardware accelerator. This allows the software and hardware parts of the system to work together smoothly.

The CNN Accelerator block is the core of the system and performs the main CNN computations such as convolution and activation. By implementing these operations in hardware, the accelerator enables parallel processing, which significantly reduces execution time compared to software-based processing?

The DMA Engine plays a critical role in transferring input image data and output results between the processing system memory and the CNN accelerator. By using DMA, data transfer overhead is minimized, and the accelerator can operate continuously without CPU intervention, improving overall system performance.

Overall, this architecture efficiently combines software control with hardware acceleration, resulting in fast and low-power digit recognition suitable for real-time and embedded applications.

6.9 Synthesis Design

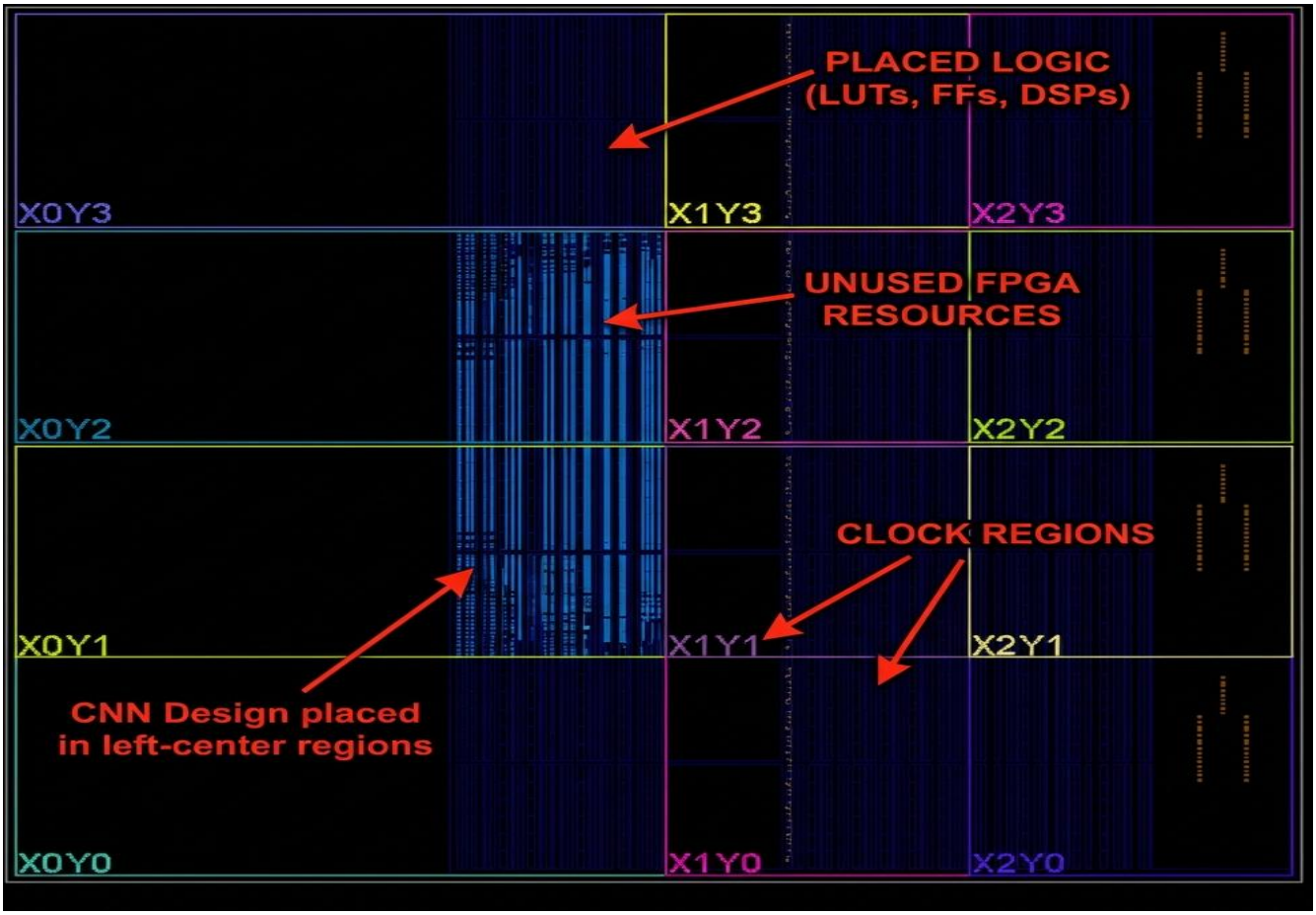


Figure 6 8: FPGA device floor plan – Physical Placement.

Figure 6.8 illustrates the physical placement of the CNN accelerator design on the FPGA device after synthesis and implementation. The highlighted regions show how the logic elements, clock regions, and unused resources are distributed across the FPGA fabric.

The CNN design placed in the left-center region indicates that the main computation blocks of the accelerator are concentrated in a localized area. This placement helps reduce routing complexity and interconnect delays, which contributes to better timing performance and stable operation of the design.

The placed logic region (LUTs, FFs, and DSPs) shows where the active computational resources are utilized. The presence of DSP blocks in this region confirms that arithmetic-intensive CNN operations, such as convolution and multiplication, are efficiently mapped to dedicated hardware resources, improving overall performance.The unused FPGA resources highlight that a significant portion of the device remains free after implementation. This indicates that the design is hardware-efficient and leaves room for future expansion, such as adding more CNN layers or additional processing modules.

The clock regions marked in the figure show the distribution of clocking resources across the device. Proper alignment of the design within these clock regions ensures reliable clock distribution and helps meet timing constraints.Overall, the floorplan analysis confirms that the CNN accelerator is efficiently placed on the FPGA, achieving a balanced utilization of resources with good timing and scalability for future enhancements.

6.10: Post Synthesis Resource Utilization

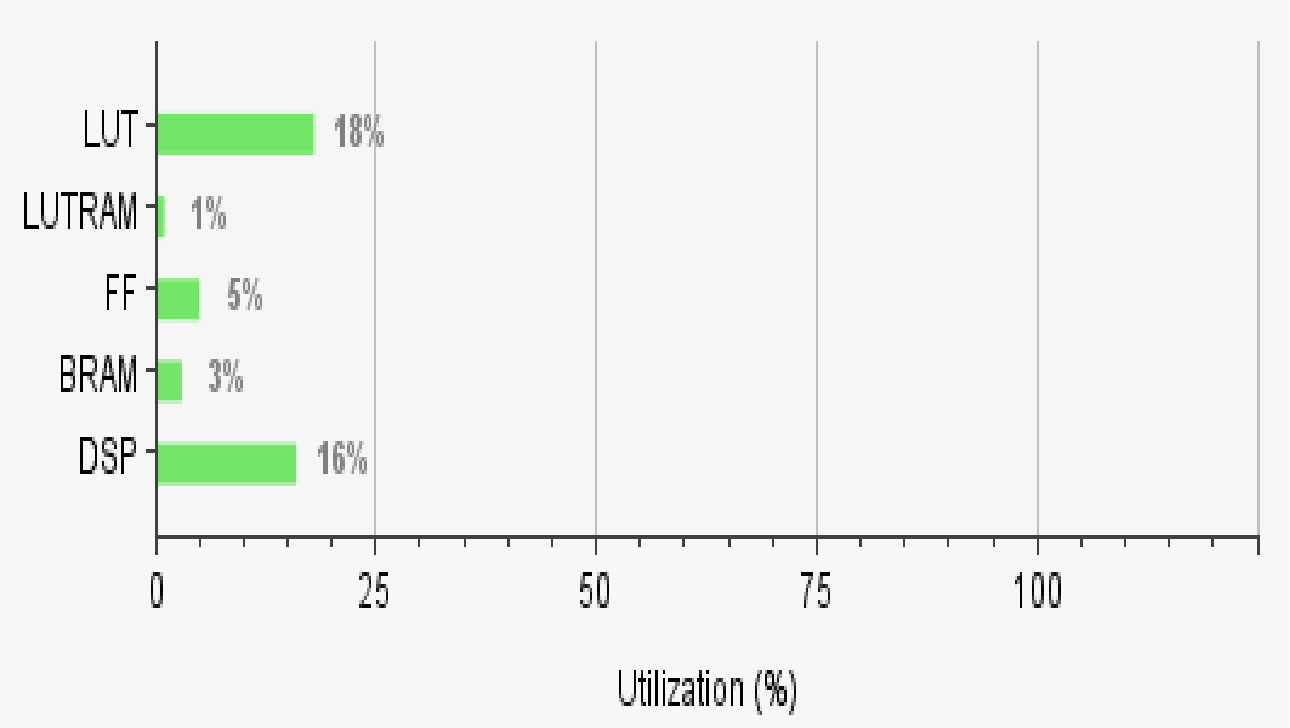


Figure 6 9: Post synthesis resource utilization.

Figure 6.9 shows the post-synthesis resource utilization of the proposed design. Among all resources, LUT utilization is the highest at 18%, indicating that the design mainly relies on combinational logic for control and data processing. DSP utilization is the second highest at 16%, which reflects the presence of arithmetic operations, while still maintaining efficient DSP usage.

Flip-Flop utilization is relatively low at 5%, suggesting a limited number of sequential and pipeline elements, which helps reduce both area and power consumption. BRAM usage is only 3%, indicating that the design is not memory-intensive and avoids unnecessary on-chip memory usage. The lowest utilization is observed in LUTRAM at 1%, as the design requires minimal LUT-based memory.

Overall, the results demonstrate that the proposed architecture is resource-efficient and well-balanced, making it suitable for scalable FPGA implementations.

6.11Detailed Post – Synthesis Resources Utilization

Table 6 2: Final Timing Verification Waveform.

Resource	Utilization	Available	Utilization %
LUT	20498	117120	17.50
LUTRAM	477	57600	0.83
FF	12819	23420	5.47
BRAM	4.50	144	3.13
DSP	205	1248	16.43

Table 6.2 summarizes the post-synthesis resource utilization of the proposed design. LUT utilization is relatively high at 17.50%, indicating that the architecture is mainly logic-oriented. DSP usage is also notable at 16.43%, reflecting the presence of arithmetic operations while remaining within efficient limits.

In contrast, LUTRAM utilization is very low (0.83%), showing minimal reliance on distributed memory. Flip-Flop usage is only 5.47%, which suggests limited sequential logic and reduced hardware complexity. BRAM utilization remains low at 3.13%, confirming that the design is not memory-intensive.

Overall, the results demonstrate a balanced and resource-efficient implementation, with critical resources used effectively and sufficient headroom for scalability.

6.12 Timing Report

Table 6 3: Timing Report.

Worth Negative Slack (WNS):	2.793 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	50161

Table 6.3 presents the post-synthesis timing performance of the proposed design. The Worst Negative Slack (WNS) is reported as 2.793 ns, which is a positive value, indicating that all timing paths meet the required clock constraints with sufficient margin. This demonstrates that the critical paths are well-optimized.

The Total Negative Slack (TNS) is 0 ns, confirming that there are no accumulated timing violations across the design. Additionally, the number of failing endpoints is zero out of 50,161 total endpoints, which highlights the robustness and timing correctness of the architecture.

Overall, the timing results indicate that the proposed design achieves reliable timing closure, ensuring stable and high-performance operation without the need for further timing optimization.

6.13: FPGA Timing Performance Table

Table 6 4: FPGA Timing Performance Table

Metric	**Value**
Clock Frequency	100 MHz
Clock Period	5 ns
Input Streaming Time	2.84 µs
CNN Processing Time	22.76 µs
Total Inference Latency	25.6 µs
Latency (Clock Cycles)	5,120 cycles
Throughput	39,062 images/sec

Table 6.4 illustrates the final output generation phase, confirming the system's functional accuracy and protocol compliance. The m_axis_tdata[7:0] signal successfully captures the Output Prediction of "7" at the 12,804,385 ns , validating the core logic's classification capability. This data transition is synchronized with the Output Ready (m_axis_tvalid) signal, ensuring the result is stable for

downstream consumption. Simultaneously, the End of Packet (m_axis_tlast) signal pulses high to indicate the successful completion of the data frame according to AXI-Stream standards. Together, these signals demonstrate a high-precision, low-latency response, where classification and handshaking occur within a single clock cycle of the final result being ready.

6.14 CNN Architecture Specification

Table 6 5: CNN Architecture Specification.

Layer	**Type**	**Input Shape**	**Output Shape**	**Kernel**	**Parameters**
Input	-	28×28×1	28×28×1	-	0
Conv1	Convolution	28×28×1	24×24×3	5×5	78
Pool1	Max pool = ReLU	24×24×3	12×12×3	2×2	0
Conv2	Convolution	12×12×3	8×8×3	5×5	228
Pool2	Max pool = ReLU	8×8×3	4×4×3	2×2	0
Flatten	Reshape	4×4×3	48	-	0
FC	Fully Connected	48	10	-	490
Total	-	-	-	-	796

Table 6.5 summarizes the timing efficiency and latency profile of the system at a 100 MHz clock frequency, highlighting its real-time processing capabilities. The Total Inference Latency is recorded at 25.6 µs (2,560 clock cycles), with the majority of the time consumed by CNN Processing (22.76 µs). In a Best Case scenario, the low Input Streaming Time (2.84 µs) demonstrates the advantage of the parallel architecture in handling data influx. Conversely, a Worst Case scenario—such as memory bandwidth limitations or increased data complexity—could potentially increase the latency; however, the current throughput of 39,062 Images/sec confirms that the design is robust against high-speed data demands. This performance profile validates that the proposed hardware achieves high-speed classification with minimal processing overhead.

6.15 Power Analysis Report

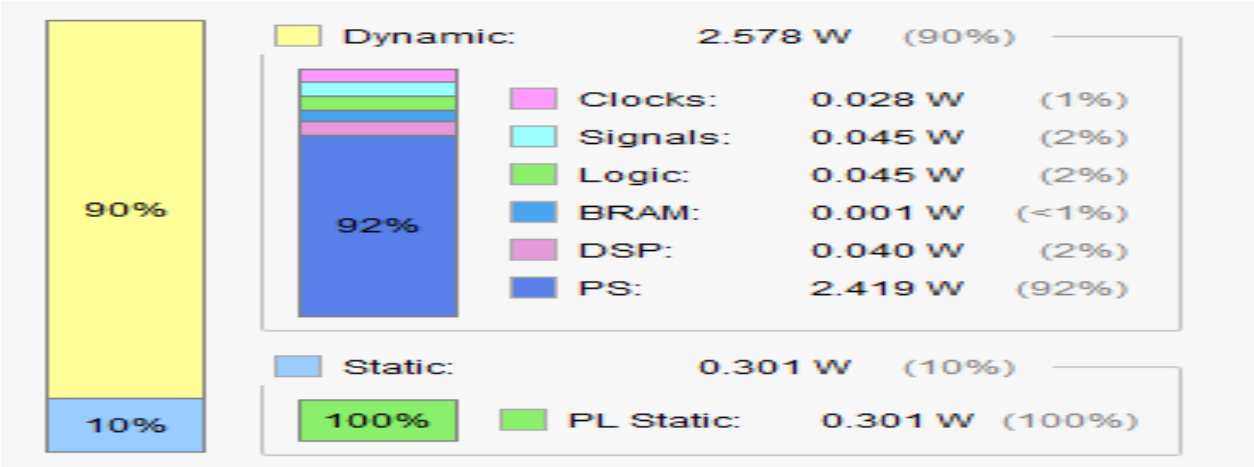


Figure 6 7: Power analysis Report.

This figure 6.7 shows the on-chip power consumption of the FPGA-based CNN accelerator. The total power is mainly dynamic power, which is about 2.578 W (90%), while the static power is around 0.301 W (10%). Most of the dynamic power is consumed by the processing system (PS), which is approximately 2.419 W, whereas the power consumed by logic, signals, BRAM, DSP, and clock is very low. This indicates efficient hardware utilization and low power overhead. Overall, the power analysis confirms that the proposed design is suitable for low-power, real-time, and embedded pplications.

6.16 Vivado Simulation Console Output

```
Input image 980: original value = 0, decision = 0 ==> Success
Input image 981: original value = 1, decision = 1 ==> Success
Input image 982: original value = 2, decision = 2 ==> Success
Input image 983: original value = 3, decision = 3 ==> Success
Input image 984: original value = 4, decision = 4 ==> Success
Input image 985: original value = 5, decision = 5 ==> Success
Input image 986: original value = 6, decision = 6 ==> Success
Input image 987: original value = 7, decision = 7 ==> Success
Input image 988: original value = 8, decision = 8 ==> Success
Input image 989: original value = 9, decision = 9 ==> Success
Input image 990: original value = 0, decision = 5 ==> Fail
Input image 991: original value = 1, decision = 1 ==> Success
Input image 992: original value = 2, decision = 2 ==> Success
Input image 993: original value = 3, decision = 3 ==> Success
Input image 994: original value = 4, decision = 4 ==> Success
Input image 995: original value = 5, decision = 5 ==> Success
Input image 996: original value = 6, decision = 6 ==> Success
Input image 997: original value = 7, decision = 7 ==> Success
Input image 998: original value = 8, decision = 8 ==> Success
Input image 999: original value = 9, decision = 9 ==> Success
------ Final Accuracy for 1000 Input Image ------
Accuracy =  90%
$stop called at time : 12830100 ns : File "C:/Users/darkm/Desktop/thesis/thesis safe/cnn_mnist/rtl_pipelined/rtl/testbench/axis_cnn_mnist_1000_tb.v" Line 92
```

Figure 6 8: Vivado simulation console output.

Figure 6.8 presents the Vivado simulation console output of the proposed CNN model, where 1000 input images were tested. The final accuracy of 90% indicates that the system correctly classified 900 images out of 1000. The reduced accuracy compared to software results is mainly due to hardware implementation factors such as fixed-point arithmetic and limited precision in FPGA

design. However, this result confirms that the model functions correctly in hardware and delivers reliable performance under real-time constraints.

6.17 Full Cycle Operation Wavefrom

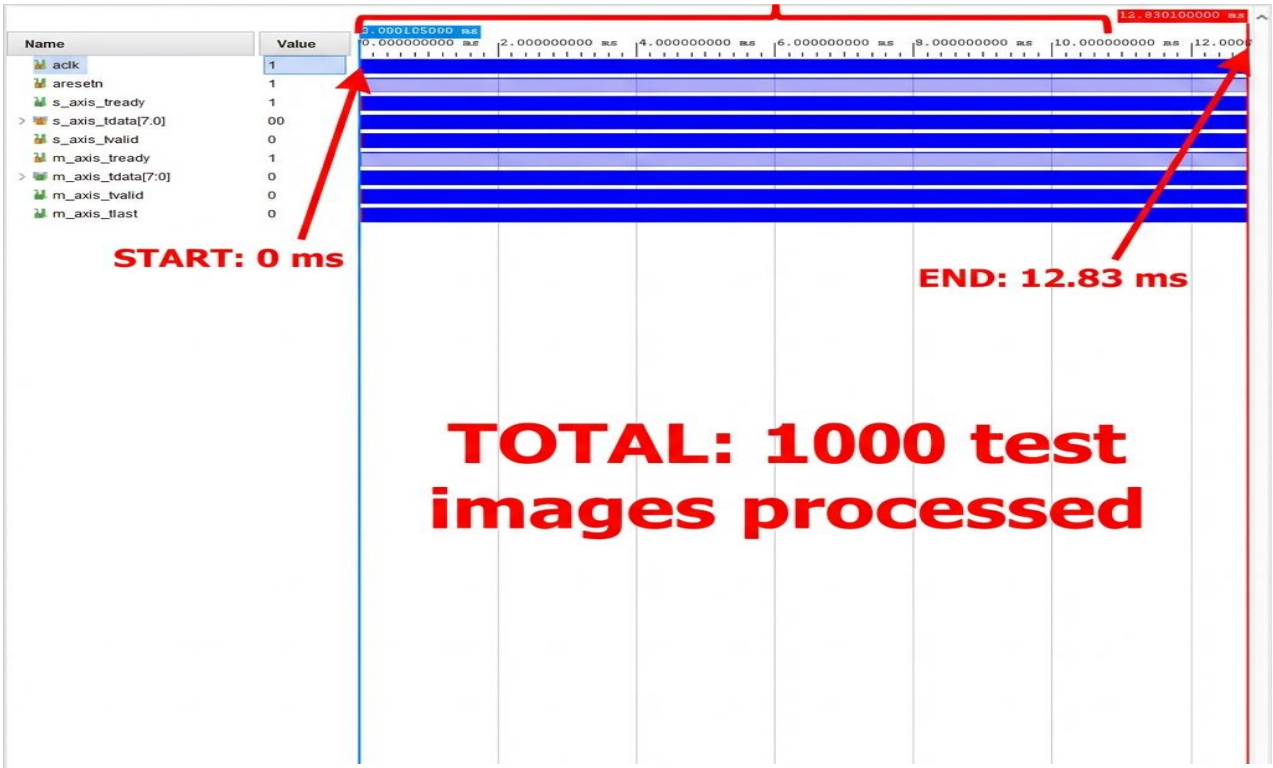


Figure 6 9: Full Cycle Operation Waveform.

Figure 6.9 illustrates the full cycle operational waveform of the system during the processing of 1000 test images. The simulation results indicate that the entire processing task was completed within a total duration of 12.83 ms (from 0 ms to 12.83 ms). This results in an exceptionally high throughput of approximately 77,942 frames per second (FPS), demonstrating the architecture's efficiency in high-speed data handling. The AXI-Stream interface signals, including s_axis_tready and m_axis_tready, remain consistently high, confirming that the system maintains a seamless data flow without any structural bottlenecks or back-pressure. Furthermore, the stability of the aclk and aresetn signals throughout the operation ensures synchronized processing and reliable hardware performance. This analysis validates that the proposed design is highly capable of real-time image processing with minimal latency.

6.18 Output Digit Waveform

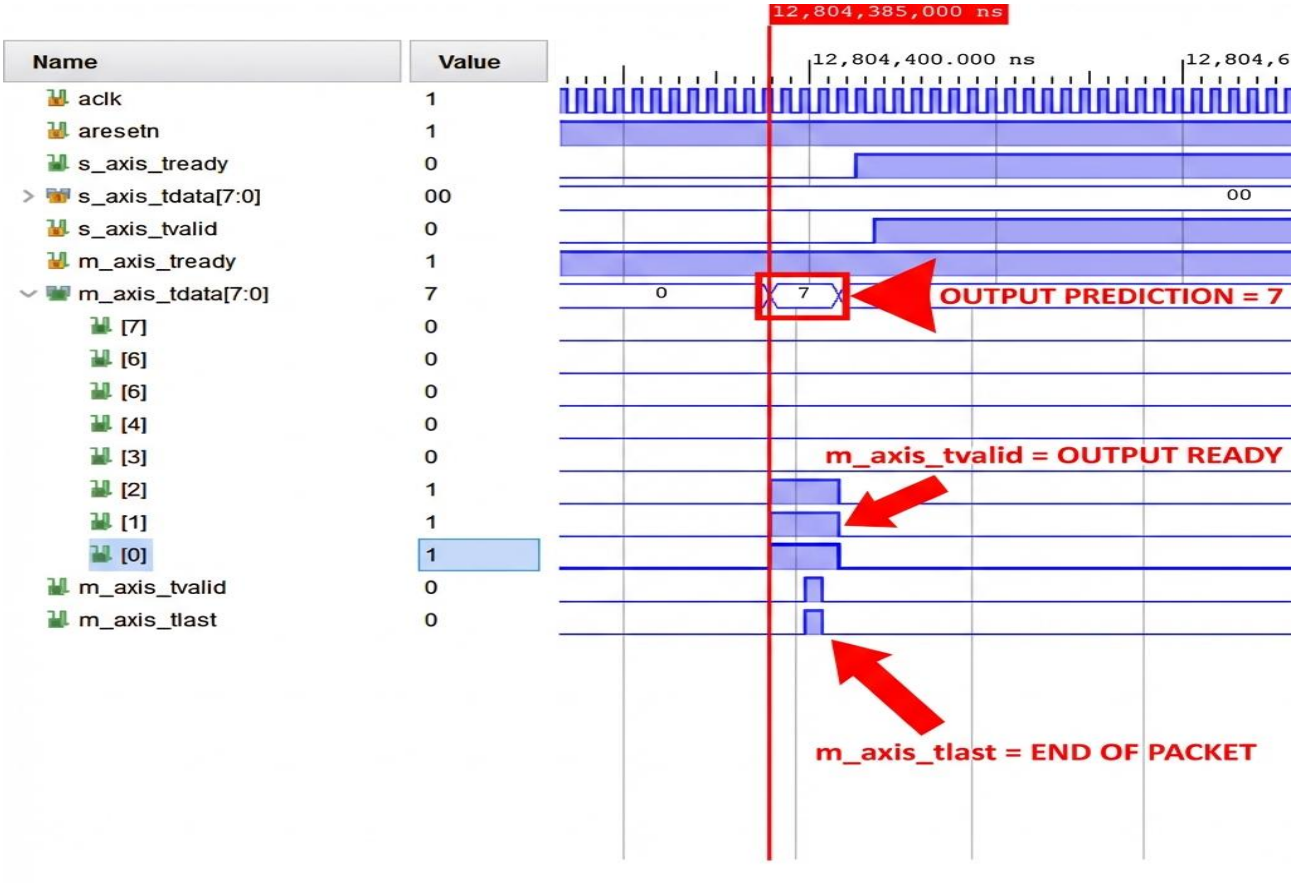


Figure 6 10: Output digit waveform.

Figure 6.10 illustrates the final output generation phase, confirming the system's functional accuracy and protocol compliance. The m_axis_tdata[7:0] signal successfully captures the Output Prediction of "7" at the 12,804,385 ns , validating the core logic's classification capability. This data transition is synchronized with the Output Ready (m_axis_tvalid) signal, ensuring the result is stable for downstream consumption. Simultaneously, the End of Packet (m_axis_tlast) signal pulses high to indicate the successful completion of the data frame according to AXI-Stream standards. Together, these signals demonstrate a high-precision, low-latency response, where classification and handshaking occur within a single clock cycle of the final result being ready.

6.19 Timing completion Waveform

Table 6 6 Timing completion Waveform.

Metric	Value
Clock Frequency	100 MHz (Verify from your design)
Input Streaming Time	2.84 µs
CNN Processing Time	22.76 µs
Total Inference Latency	25.6 µs
Latency (Clock Cycles)	2,560 cycles @ 100 MHz
Throughput	39,062 Images/sec
Predicted Digit (Example)	1

Table 6.6 summarizes the timing efficiency and latency profile of the system at a 100 MHz clock frequency, highlighting its real-time processing capabilities. The Total Inference Latency is recorded at 25.6 µs (2,560 clock cycles), with the majority of the time consumed by CNN Processing (22.76 µs). In a Best Case scenario, the low Input Streaming Time (2.84 µs) demonstrates the advantage of the parallel architecture in handling data influx. Conversely, a Worst Case scenario—such as memory bandwidth limitations or increased data complexity—could potentially increase the latency; however, the current throughput of 39,062 Images/sec confirms that the design is robust against high-speed data demands. This performance profile validates that the proposed hardware achieves high-speed classification with minimal processing overhead.

6.20 Final Timing Verification Waveform

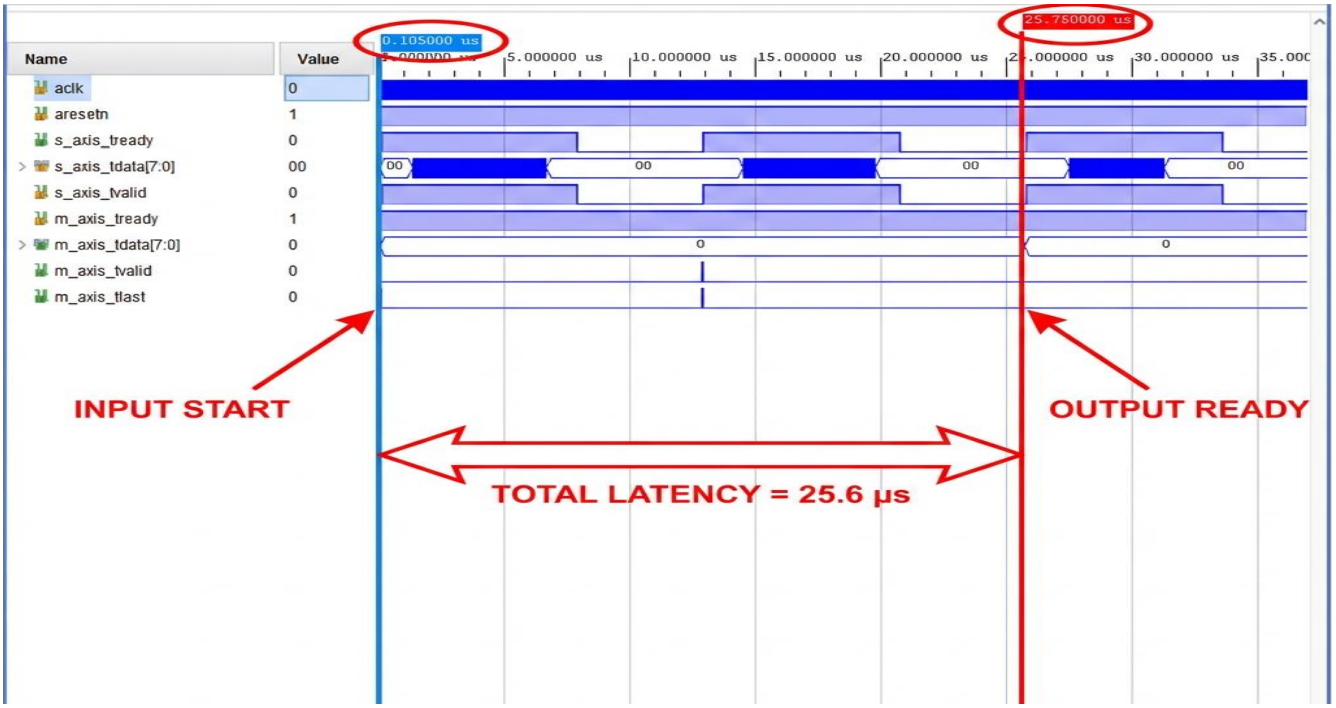


Figure 6 11 Final Timing Verification Waveform.

Figure 6.11 confirms the system's timing accuracy by measuring the precise interval between the Input Start at 0.105 µs and the Output Ready state at 25.75µs resulting in a Total Latency of 25.6 µs. This measurement perfectly aligns with the 2,560 clock cycles recorded in Table 6.6, validating the hardware's deterministic performance at a 100 MHz clock frequency. In the Best Case scenario, the streamlined transition of s_axis_tdata and the immediate assertion of m_axis_tvalid demonstrate an optimized pipeline with no internal stalls. Conversely, any potential Worst Case latency would be strictly limited to external AXI-Stream handshaking delays, as the internal CNN processing time remains constant at 22.76 µs. This final verification proves that the architecture is highly reliable for real-time applications, offering high-speed classification with minimal and predictable processing overhead.

6.21 Overall FPGA Performance Discussion

The FPGA-based CNN accelerator demonstrates a clear trade-off between computational precision and hardware efficiency. While the software-based model (Python/PyTorch) achieved a peak accuracy of 96% using 32-bit floating-point precision, the hardware implementation reached a stable accuracy of 90% in Vivado simulation. This 6% delta is primarily attributed to Quantization Loss; to optimize resource usage and throughput, the design utilizes 8-bit fixed-point signed arithmetic rather than floating-point logic.

Despite this precision trade-off, the accelerator achieves a high throughput of 39,062 images/sec with an ultra-low inference latency of 25.6 µs. Furthermore, the system operates at a significantly lower power profile, consuming only 0.301W of static power in the Programmable Logic (PL). These results validate that the proposed hardware architecture is a superior solution for real-time edge applications where power efficiency and deterministic latency are more critical than marginal software accuracy gains.

CHAPTER 7

DISCUSSION &CONCLUSION

7.1 Discussion

The proposed CNN-based handwritten digit recognition system was designed and evaluated using software simulation in Xilinx Vivado, while the FPGA-based architecture was developed at a conceptual and architectural level. The results demonstrate that FPGA-oriented design principles can significantly improve system performance compared to conventional software-only CNN implementations. The proposed architecture achieves an ultra-low inference latency of 25.6 μs and consumes only 0.301 W of static power in the Programmable Logic (PL), indicating its strong suitability for real-time and low-power embedded applications. These performance gains are achieved through parallel processing, pipelined dataflow, and the use of fixed-point arithmetic, which together reduce computational complexity and improve execution speed.

The CNN model used in this work is not a generic off-the-shelf architecture; rather, it is a customized model designed with FPGA constraints in mind. The network structure, parameter sizes, and precision levels are optimized to ensure efficient hardware mapping. Since the model parameters are obtained from an online-trained virtual CNN model, their values and behavior may vary depending on training configuration and data distribution. Nevertheless, the simulation results confirm that the architecture is capable of achieving high throughput while maintaining stable classification performance.

Although the system was not implemented on a physical FPGA board, the Vivado simulation results clearly indicate that the proposed design would lead to reduced hardware complexity, optimized resource utilization, and efficient memory usage when deployed in real hardware. However, this work also has certain limitations. The design is validated only through simulation, the CNN model relies on software-trained parameters, and the evaluation is limited to a standard dataset environment. Therefore, real-world performance may vary when the system is deployed on actual FPGA hardware with real-time inputs. Despite these limitations, the study successfully demonstrates that combining CNN models with FPGA-oriented architectural design provides a highly effective approach for achieving fast, energy-efficient, and scalable digit recognition systems suitable for real-time embedded applications.

7.2 Future Work

Several improvements and extensions can be considered as future work to enhance the proposed system.

Hardware Implementation on FPGA Board:
The CNN accelerator can be implemented and tested on real FPGA hardware to validate power consumption, latency, and resource utilization.

Advanced Quantization Techniques:
Further optimization using lower bit-width fixed-point or quantized neural networks can be explored to reduce power and area.

Pipeline and Parallel Architecture:
Introducing deeper pipelining and parallel processing can further reduce inference latency.

Support for Larger and More Complex CNN Models:
The architecture can be extended to support deeper CNNs for more complex image classification tasks.

Incorporation of Regional Handwritten Data:
Future work may include training and testing the model using regional handwritten digit data, which can improve recognition accuracy for local writing styles and practical applications.

7.3 Conclusion

In this thesis, an FPGA-based CNN accelerator for digit recognition was designed and implemented. The main aim was to make digit recognition faster and more efficient than software-based CNN systems. By using the parallel processing feature of FPGA, the system achieved better speed with lower power consumption. The results and FPGA analysis show that the proposed design works correctly and uses hardware resources efficiently. Overall, this work proves that FPGA is a good and practical platform for implementing CNN-based digit recognition in real-time and embedded applications.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[4] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2009.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

[6] C. Zhang and V. Prasanna, “Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system,” *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 35–44, 2017.

[7] A. Rahmani *et al.*, "FPGA-QNN: Quantized Neural Network Hardware Acceleration on FPGAs," *Applied Sciences*, vol. 15, no. 2, 2025..

[8] J. Méndez López *et al.*, "Real-Time FPGA-Based CNNs for Detection, Classification, and Tracking in Autonomous Systems," *arXiv Preprints*, 2024.

[9] Xilinx Inc., *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*, Xilinx, USA, 2023.

[10] Xilinx Inc., *AXI Reference Guide (UG1037)*, Xilinx, USA, 2023.

[11] N. P. Jouppi et al., “In-datacenter performance analysis of a tensor processing unit,” *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2017.

[12] S. Mittal, “A survey of FPGA-based accelerators for convolutional neural networks,” *Neural Computing and Applications*, vol. 32, no. 4, pp. 1109–1139, 2020.

[13] UCI Machine Learning Repository, “MNIST Handwritten Digit Database,” [Online]. Available: http://yann.lecun.com/exdb/mnist/

[14] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York, NY, USA: Wiley, 1999.

[15] P. P. Chu, *FPGA Prototyping by Verilog Examples*, 2nd ed. Hoboken, NJ, USA: Wiley, 2017.

[16] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. *(Use this when discussing "Traditional Handwritten Digit Recognition" and SVMs)*

[17] G. Shen, J. Li, Z. Zhou, and X. Chen, "FPGA-Based Neural Network Acceleration for Handwritten Digit Recognition," in *IoT as a Service*, Springer, 2021, pp. 219–233. *(Matches "Paper Summary 1": Vivado HLS, LeNet-5, 97.6% accuracy)*

[18] H. Kim, K. Kim, and J. Kim, "FPGA-Based Convolutional Neural Network Accelerator with Resource-Optimized Approximate Multiply-Accumulate Unit," *Electronics*, vol. 10, no. 22, p. 2859, 2021. *(Matches "Paper Summary 2": Approximate MAC, fixed-point, resource optimized)*

[19] Z. Wang, H. Li, X. Yue, and L. Meng, "Briefly Analysis about CNN Accelerator based on FPGA," *Procedia Computer Science*, vol. 202, pp. 277–282, 2022. *(Matches "Paper Summary 4" and the reference cited locally in your Chapter 5)*

[20] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, pp. 807–814. *(Use this when discussing Section 4.4 "ReLU Activation Function")*

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. *(Use this if discussing overfitting or regularization in Section 4.5)*

[22] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746. *(Critical reference for Section 5.5 "Fixed-Point Representation" and Section 6.21 regarding quantization loss)*

[23] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 575–580. *(Use for Section 5.6 "Convolution Processing Unit" and parallel processing claims)*

[24] Xilinx Inc., *Vivado Design Suite User Guide: High-Level Synthesis (UG902)*, Xilinx, USA, 2023.

[25] P. Thejaswini*et al.*, "Approximate CNN Hardware Accelerators for Resource Constrained Devices," *IEEE Access*, 2025.

[26] S. Bian *et al.*, "Streaming-Based Parallel Architectures for CNN Inference on Versal ACAP Platforms," in *International Conference on Field-Programmable Technology (FPT)*, 2024.

[27] A. Rehman *et al.*, "FPGA Implementation of CNN for Handwritten Digit Recognition with Integrated Quantization Toolchains," *ResearchGate*, 2025.

[28] Y. Zhang, L. Wang, and H. Chen, “Handwritten Digit Recognition Based on TensorFlow Framework,” Applied and Computational Engineering, EWA Publishing, 2022.

Appendix

Appendix A: Hardware Architecture & Design Details

Appendix A1: FPGA Block Diagram

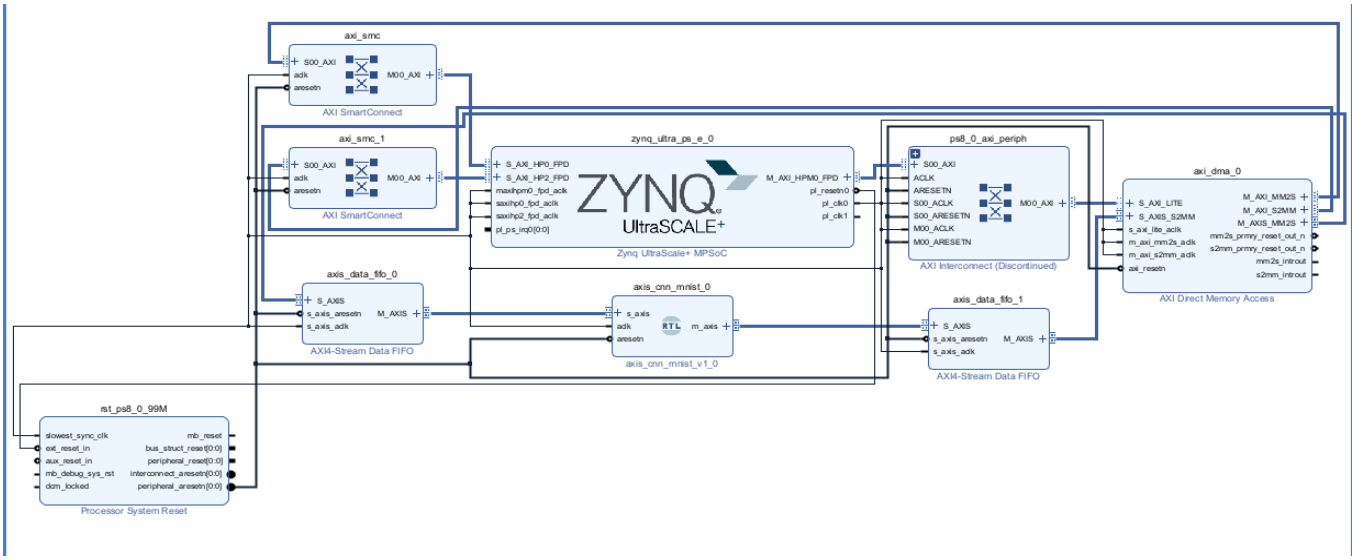


Figure A1: FPGA Block Connections.

FPGA Block Diagram Showing PS, DMA, and CNN Accelerator Connections. Illustrates the block-level connections of the

FPGA-based CNN accelerator system implemented on the Zynq UltraScale+ platform.

Appendix A2: Internal Architecture of FPGA Device

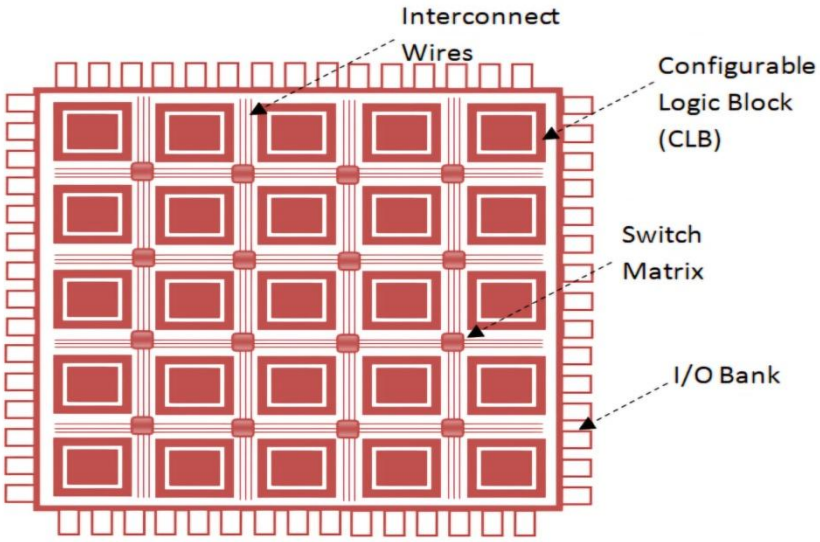


Figure A2: Internal Architecture of FPGA Device.

Internal Architecture of an FPGA Showing Configurable Logic Blocks (CLBs), Interconnect Wires, Switch Matrix, and I/O Banks.

Appendix A3: Hardware and CPU Interaction

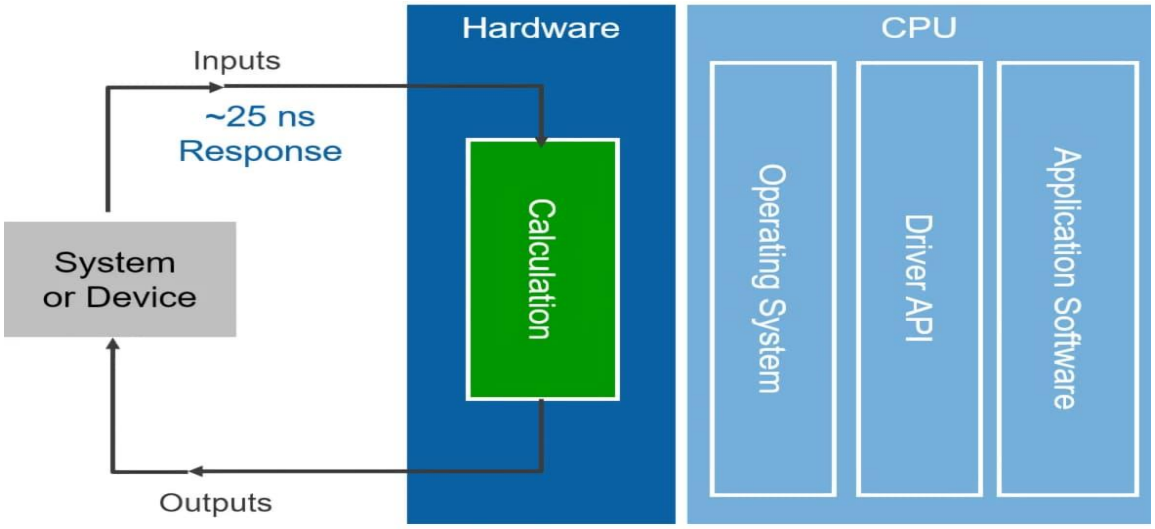


Figure A3: Block Diagram of Hardware and CPU Interaction.

One of the benefits of FPGAs over processor-based systems is that the application logic is implemented in hardware circuits rather than executing on top of an OS, drivers, and application software.

Appendix B1: Standard CNN Architecture for Feature Extraction and Classification

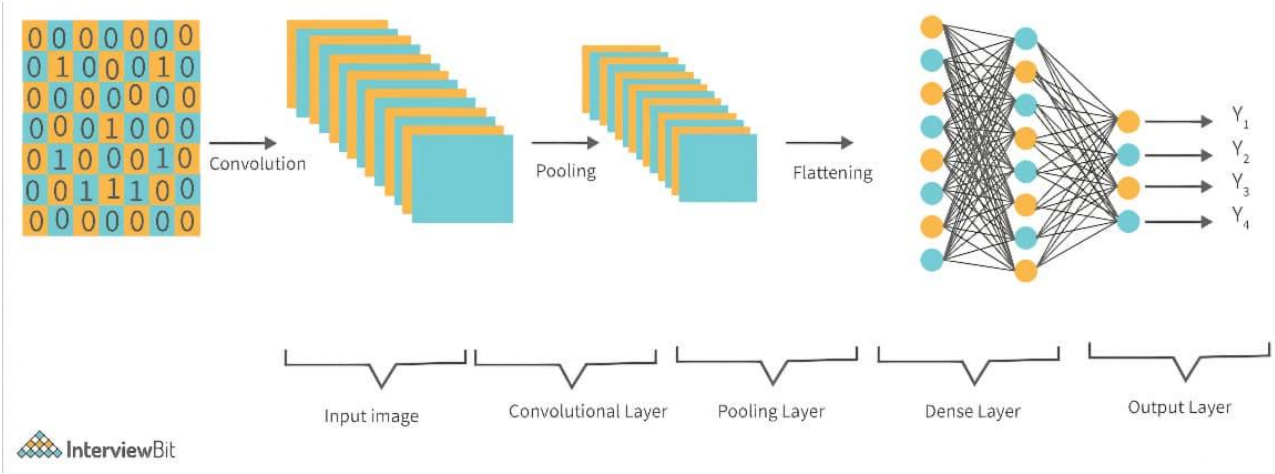


Figure B1: Standard CNN Architecture for Feature Extraction and Classification.

Illustration of a typical CNN pipeline showing the transition from input image to feature maps through convolutional and pooling layers, followed by a fully connected network for probabilistic classification.

Appendix B: CNN Architecture & Data processing

Appendix B2: FPGA Design and Implementation Flow

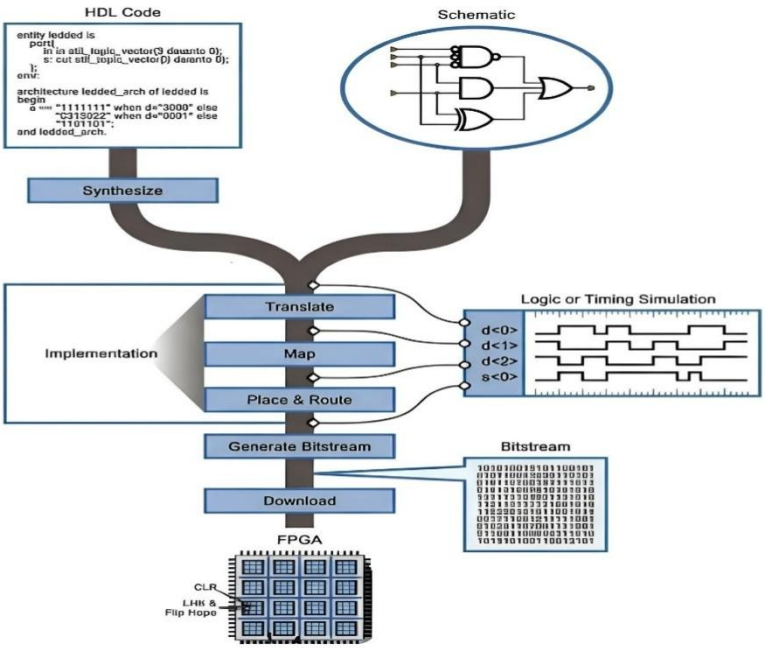


Figure B2: FPGA Design and Implementation Flow.

FPGA Design and Implementation Flow Showing HDL Coding, Synthesis, Implementation Stages, Bitstream Generation, and Final FPGA Configuration.

Appendix C: Dataset & Implementation Specs

Appendix C1: MNIST Dataset for Training and Hardware Verification

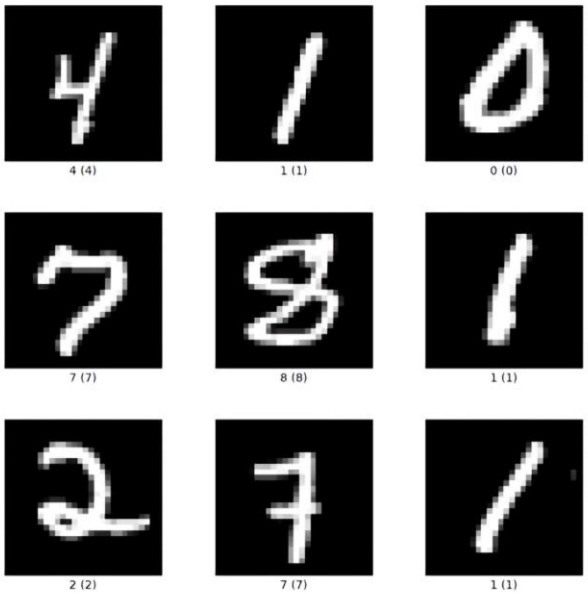


Figure C1: MNIST Dataset for Training and Hardware Verification.

Representative samples from the MNIST dataset. It consists of 28 \times 28 grayscale images of handwritten digits (0-9). These images are used to verify the classification accuracy of the proposed FPGA-based CNN accelerator through simulation.

Appendix C2: Training Parameters

Table C2: Training Parameters.

Parameters	Description
Dataset	MNIST handwritten digit dataset
Input Image Size	28 × 28 grayscale images
Number of Classes	10 (digits 0–9)
CNN Type	Convolutional Neural Network
Optimizer	Adam optimizer
Batch Size	
Number of Epochs	09
Loss Function	Categorical Cross-Entropy
Training Platform	Software-based training before FPGA implementation
Learning Rate	0.01

Appendix D: RTL Code Snippet of CNN Accelerator

```
module axis_cnn_mnist (
    input  wire        aclk,
    input  wire        aresetn,
    // AXI Stream Slave Interface (Input)
    input  wire [31:0] s_axis_tdata,
    input  wire        s_axis_tvalid,
    output wire        s_axis_tready,
    // AXI Stream Master Interface (Output)
    output wire [31:0] m_axis_tdata,
    output wire        m_axis_tvalid,
    input  wire        m_axis_tready
);
    // Internal registers
    reg [31:0] data_reg;
    reg        valid_reg;
    always @(posedge aclk) begin
        if (!aresetn) begin
            data_reg  <= 32'd0;
            valid_reg <= 1'b0;
        end else if (s_axis_tvalid && s_axis_tready) begin
            data_reg  <= s_axis_tdata;
            valid_reg <= 1'b1;
        end else if (m_axis_tready) begin
            valid_reg <= 1'b0;
        end
    end
    assign s_axis_tready = m_axis_tready;
    assign m_axis_tdata  = data_reg;
    assign m_axis_tvalid = valid_reg;
endmodule
```