Highly Efficient Hand-Written Digit Recognition using Multilayer Perceptrons through ZyNet and its Hardware Implementation

A

Project submitted

For partial fulfilment of the requirements for the degree of

Bachelor of Technology

in

ELECTRONICS AND TELECOMMUNICATION ENGINEERING

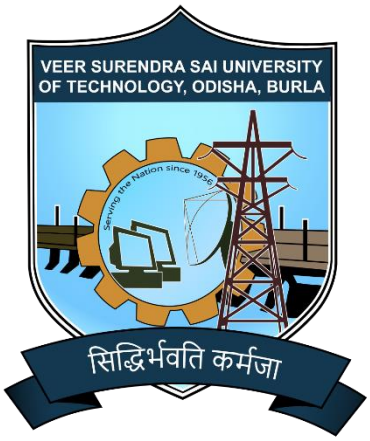
Submitted by

SAMPAD MOHANTY

Registration No.: 2002070059

Under the guidance of

Mr. Aditya Kumar Hota



DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, ODISHA

SIDDHI VIHAR, BURLA, SAMBALPUR-768018, ODISHA, INDIA

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA, ODISHA

CERTIFICATE

This is to certify that the work contained in this report of project entitled **"Highly Efficient Hand-Written Digit Recognition using Multilayer Perceptrons through ZyNet and its Hardware Implementation"** submitted by **Sampad Mohanty (2002070059)** for the award of the degree of **Bachelor of Technology** in **Electronics and Telecommunication Engineering** at **Veer Surendra Sai University of Technology, Burla,** is a record of bonafide project works carried out by him under my direct super vision and guidance.

I considered that the project has reached the standards and fulfilling the requirements of the rules and regulations relating to the nature of the degree. The contents embodied in this project report have not been submitted for the award of any other degree or diploma in this or any other university.

Mr. Aditya Kumar Hota

Assistant Professor
Department of Electronics and
Telecommunication Engineering
VSSUT, Burla

Prof. Harish Kumar Sahoo

Head of Department
Department of Electronics and
Telecommunication Engineering
VSSUT, Burla

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA, ODISHA

CERTIFICATE OF APPROVAL

This is to certify that the work contained in this report of project entitled **"Highly Efficient Hand-Written Digit Recognition using Multilayer Perceptrons through ZyNet and its Hardware Implementation"** submitted by **Sampad Mohanty (2002070059)** under the guidance and kind support of **Mr. Aditya Kumar Hota** to VSSUT, Burla has been examined by us and is approved forthe degree of Bachelor of Technology in Electronics and Telecommunication Engineering.

(INTERNAL EXAMINER) (EXTERNAL EXAMINER)

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA, ODISHA

DECLARATION

I hereby declare that this written submission on the project entitled **"Highly Efficient Hand-Written Digit Recognition using Multilayer Perceptrons through ZyNet and its Hardware Implementation"** represents my innovation in my own words, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will cause disciplinary action by the university and can also invoke penal action from the sources which thus have not properly cited or from whom proper permission has not been taken when needed.

Date:

Place:

Sampad Mohanty

Regd No. 2002070059

ACKNOWLEDGEMENT

The success and outcome of this major project required a lot of guidance and assistance, and I am extremely privileged to have had this all through the completion of my project. All that I have done is only due to such supervision and assistance and I should not forget to thank them. I heartily thank Mr. Aditya Kumar Hota (Assistant Professor, Department of Electronics and Telecommunication Engineering, VSSUT, Burla) for his excellent guidance and giving me an opportunity to learn and make this major project report successfully. His enormous trust and continuous support inspired me in the most crucial moments of making the right decision and I am feeling blessed to work under him. I respect and thank Prof. Harish Kumar Sahoo (Professor and Head of the Department, Department of Electronics and Telecommunication Engineering, VSSUT, Burla) for providing me an opportunity for a presentation as a part of final year major project evaluation and giving me all support and guidance. I am extremely thankful to him for providing such a nice support and guidance.

Sampad Mohanty (2002070059)

Abstract

Multilayer Perceptrons (MLPs) constitute a foundational element in artificial intelligence applications, spanning diverse domains from image classification to natural language processing. However, optimizing their hardware implementation presents challenges due to the intricate neural network architecture and hardware constraints. This study investigates ZyNet, an innovative framework tailored specifically for FPGA-based deployment, with the aim of addressing these challenges. Through a comprehensive review of existing literature, prevalent inefficiencies in conventional approaches are identified, providing a foundational basis for ZyNet exploration. ZyNet offers a systematic methodology to utilize the inherent capabilities of FPGAs, optimizing resource utilization and enhancing performance. By harnessing ZyNet's architectural insights, this project designs and synthesizes a bespoke MLP implementation tailored for the Zedboard Zynq7000 platform. The resulting system showcases notable improvements in accuracy and resource utilization compared to traditional methodologies, underscoring the transformative potential of ZyNet in redefining FPGA-based AI deployments. This endeavor contributes to the advancement of hardware-accelerated neural networks, facilitating more efficient and scalable AI applications across diverse domains.

Keywords: Artificial intelligence, Multilayer Perceptrons, ZyNet framework, FPGA implementation, Neural network optimization, Hardware acceleration.

Contents

List of Figures and Tables

Chapter

1

Introduction

Artificial Intelligence (AI) stands as a transformative force reshaping various sectors, from healthcare to finance, with its ability to simulate human intelligence in machines. At the heart of AI lies neural networks, computational models inspired by the human brain's interconnected network of neurons. These networks, composed of layers of artificial neurons, possess the remarkable capacity to learn from data, recognize patterns, and make decisions without explicit programming. One of the most promising domains benefiting from AI and neural networks is image processing. By leveraging neural networks, AI algorithms can analyze, interpret, and manipulate visual data with unprecedented accuracy and efficiency. Applications span diverse areas, including medical imaging for disease diagnosis, surveillance systems for security, and autonomous vehicles for object detection and recognition [1], [2], [3], [4], [5], [6], [7].

In image processing, neural networks play a pivotal role in extracting meaningful information from visual data. They enable tasks such as image classification, object detection, segmentation, and image generation, revolutionizing industries reliant on visual information. Through the training process, neural networks learn to discern intricate patterns and features within images, enabling them to identify objects, recognize faces, and even generate realistic images. With advancements in deep learning, convolutional neural networks (CNNs) have emerged as the cornerstone of image processing tasks, demonstrating unparalleled performance in tasks like image recognition and object detection. However, alongside CNNs, Multilayer Perceptrons (MLPs) remain a foundational architecture, particularly in scenarios where interpretability and simplicity are paramount. Understanding the underlying principles and capabilities of MLPs is essential for unlocking their potential in various image processing applications.

1.1 Introduction of MLP

Artificial Neural Networks (ANNs) are computational models inspired by the structure and function of the human brain, capable of learning from data to perform a diverse range of tasks. In image processing, ANNs have revolutionized fields such as computer vision and pattern recognition by automatically extracting features and recognizing patterns from visual data. Among these architectures, the Multi-Layer Perceptron (MLP) stands out as a fundamental model widely used for image classification, object detection, and image segmentation [8], [9], [10], [11], [12], [13]. Its simple architecture and low resource utilization make MLPs particularly suitable for hardware implementation, enabling faster processing compared to other neural network architectures. The feasibility of implementing MLPs in hardware has garnered significant interest due to their potential for real-time and energy-efficient image processing tasks. While hardware implementation introduces challenges such as limited resources and computational constraints, advancements in FPGA and ASIC technologies have made it increasingly feasible to deploy MLPs directly onto hardware platforms, enabling efficient parallel processing and accelerated inference for image processing applications. Several frameworks are available to describe the model in Hardware Description Language (HDL), with ZyNet being a user-friendly option chosen for its ease of use in the implementation.

1.2 Introduction to ZyNet Framework

ZyNet [14] represents a sophisticated hardware description language (HDL) framework meticulously crafted for deploying neural network models onto FPGA (Field Programmable Gate Array) platforms. With its array of tools and libraries, ZyNet simplifies the design, synthesis, and optimization process for hardware implementation, prioritizing user-friendliness and efficiency. By abstracting away FPGA programming intricacies and offering high-level constructs, ZyNet empowers designers to swiftly translate neural network models into optimized hardware circuits. Its modular architecture facilitates seamless integration of custom architectures and optimization strategies, enabling rapid prototyping and experimentation. ZyNet harnesses the parallel processing capabilities of FPGAs, accelerating neural network inference tasks while ensuring optimal resource utilization and power efficiency. As a pivotal tool for leveraging FPGA technology in neural network deployment, ZyNet offers both researchers and practitioners a comprehensive solution for hardware-based AI applications.

1.3 Motivation

The motivation behind this endeavor stems from the burgeoning demand for efficient and high-performance hardware implementations of neural network models, particularly in the realm of artificial intelligence (AI) and machine learning (ML) applications. Traditional software-based approaches to neural network inference tasks often encounter performance bottlenecks when faced with large-scale datasets or real-time processing requirements. As AI continues to permeate various domains, including autonomous vehicles, robotics, healthcare, and IoT devices, there is a pressing need for hardware accelerators capable of delivering real-time inferencing while maintaining low power consumption and cost-effectiveness.

Field Programmable Gate Arrays (FPGAs) offer a compelling solution to this challenge by providing customizable hardware platforms that can be tailored to the specific requirements of neural network models. Unlike application-specific integrated circuits (ASICs), FPGAs offer flexibility in design iteration and rapid prototyping, making them well-suited for research and development efforts in the field of hardware-accelerated AI. Additionally, FPGAs boast parallel processing capabilities that align with the inherently parallel nature of neural network computations, enabling efficient utilization of resources and accelerated inference speeds. This potential for parallelism, coupled with the reconfigurability of FPGA architectures, presents a compelling opportunity to deploy neural network models directly onto hardware, bypassing the latency and resource constraints associated with software-based implementations.

1.4 Objective

The objective of this research initiative is to develop and deploy a hardware-accelerated Multilayer Perceptron (MLP) model on an FPGA platform to address the increasing demand for efficient and high-performance neural network implementations in embedded systems, edge computing, and real-time applications. By harnessing FPGA parallel processing capabilities and reconfigurability, the aim is to explore novel methodologies for executing MLP computations directly in hardware, overcoming computational bottlenecks associated with traditional software-based approaches. This involves optimizing execution speed and resource utilization to enable real-time inference capabilities with minimal latency, balancing computational efficiency, accuracy, and hardware constraints. Additionally, the research seeks

to contribute to the discourse on FPGA-based AI acceleration by evaluating and validating the hardware-accelerated MLP model's efficacy across real-world use cases and application scenarios, benchmarking its performance against software-based counterparts and scrutinizing scalability, power efficiency, and adaptability. Ultimately, the goal is to advance the understanding of FPGA-based AI acceleration and facilitate the widespread adoption of hardware-accelerated neural network solutions across diverse domains and industries.

1.5 Thesis Organization and Chapter Wise Contribution

Chapter 1: In the introduction, the project's objectives and scope are outlined, emphasizing the significance of optimizing MLP implementations using ZyNet.

Chapter 2: The literature review meticulously surveys existing studies, identifying gaps and challenges in current approaches while providing a foundation for the proposed methodology.

Chapter 3: Methodology chapter elaborates on ZyNet's architecture, detailing its implementation process and highlighting its advantages over existing methods.

Chapter 4: The conclusions section succinctly summarizes key findings and reflects on ZyNet's achievements and limitations, offering insights into future research directions. Future scopes explore potential avenues for enhancing ZyNet and advancing FPGA-based deployment of neural networks, paving the way for further innovation in the field.

Chapter

2

Literature Survey

2.1 Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron (MLP) stands as a pivotal architecture in the realm of artificial neural networks (ANNs), recognized for its ability to discern intricate data patterns. Unlike its predecessor, the perceptron, the MLP encompasses multiple interconnected layers of neurons, enabling it to identify non-linear relationships and undertake complex tasks such as classification and regression. By incorporating hidden layers between input and output layers,

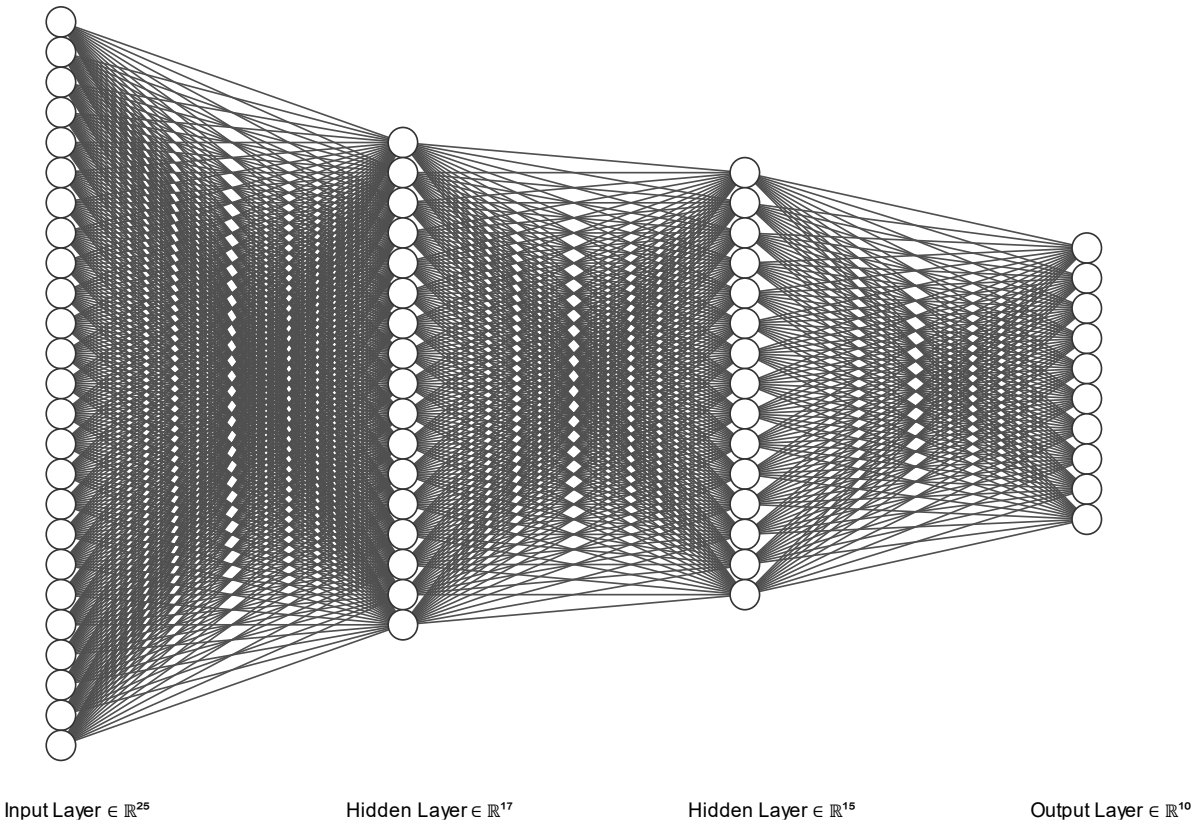


Figure 2.1: Multilayer Perceptron used for Handwritten Digit Classification

MLPs can extract hierarchical features from data, facilitating the modeling of complex data distributions and adaptation to diverse problem domains. This adaptability has propelled MLPs to prominence across various domains, including image and speech recognition, natural language processing, and financial forecasting, where they excel at tasks requiring pattern recognition, prediction, and decision-making.

2.2 Architecture of MLP

The architecture of a Multi-Layer Perceptron (MLP), as illustrated in figure 2.1, comprises interconnected layers of neurons, each layer serving a distinct purpose in information processing. At the forefront is the input layer, where raw data is received and represented by individual neurons, with each neuron corresponding to a specific feature of the input data. The input layer acts as the initial interface between the external data and the neural network, transmitting the information to the subsequent layers for further processing. Following the input layer are one or more hidden layers, where the bulk of computation occurs. Each neuron in a hidden layer is connected to every neuron in the preceding layer, forming a fully connected network. Through these connections, weighted sums of input data are computed, followed by the application of activation functions that introduce non-linearities to the network's transformations. These non-linear activations enable the network to capture complex patterns and relationships within the data, enhancing its capacity for learning and generalization.

Within each neuron of a Multi-Layer Perceptron (MLP), several operations, as depicted in figure 2.2 take place to process incoming information and generate an output signal. At its core, a neuron computes a weighted sum of its inputs, where each input is multiplied by a corresponding weight parameter. These weighted sums are then aggregated, and a bias term may be added to the result. The purpose of the bias is to allow the neuron to learn an optimal threshold for activation, influencing the neuron's responsiveness to different input patterns. Once the weighted sum and bias are computed, an activation function is applied to introduce

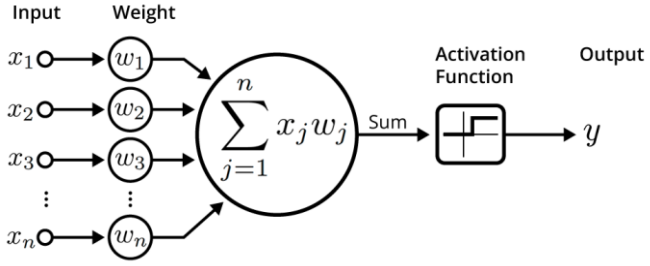


Figure 2.2: Artificial Neuron in MLP

non-linear transformations to the neuron's output. Activation functions like the sigmoid, ReLU (Rectified Linear Unit), or tanh (Hyperbolic Tangent) serve to introduce non-linearities into the network, enabling it to learn complex relationships within the data. This non-linearity is crucial for the network's ability to model non-linear patterns and make accurate predictions. Overall, within each neuron, a sequence of operations involving weighted sum computation, bias addition, and activation function application occurs, allowing the neuron to transform input signals into meaningful output representations.

The final layer of the MLP is the output layer, where the processed information is transformed into the desired format for the given task. The number of neurons in the output layer depends on the nature of the problem being addressed, with each neuron typically representing a class label or a regression value. During inference, the output layer produces predictions or estimations based on the learned patterns encoded in the network's parameters. Overall, the architecture of an MLP is characterized by its flexibility and scalability, allowing for the configuration of various network architectures tailored to specific applications. By adjusting the number of neurons in each layer and the depth of the network, MLPs can adapt to diverse datasets and tasks, making them versatile tools for solving a wide range of machine learning problems.

2.3 Activation Functions

Activation functions are essential components of neural networks, dictating the behavior of individual neurons and shaping the overall performance of the network. Among the commonly used activation functions is the sigmoid function, recognized for its characteristic S-shaped curve, as in figure 2.3(a), that maps input values to a range between 0 and 1. This function finds

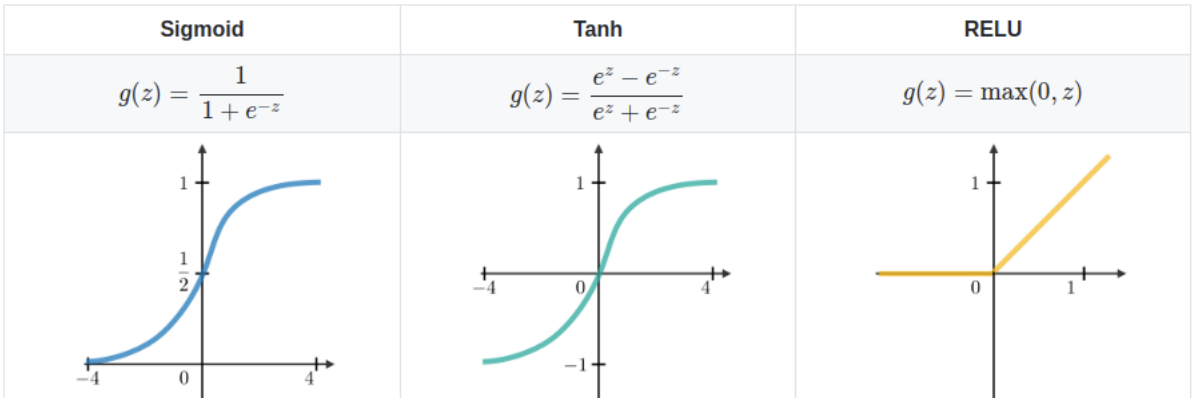


Figure 2.3: Mathematical and graphical representation of (a) Sigmoid, (b) Tanh and (c) ReLU activation functions

widespread application in binary classification tasks, where it interprets the output as probabilities. However, one notable drawback of sigmoid functions is the vanishing gradient problem, especially evident in deep neural networks with numerous layers. As inputs move away from the origin, gradients approach zero, impeding the learning process during backpropagation and hindering the convergence of the network.

Rectified Linear Units (ReLU) have emerged as a popular alternative to sigmoid functions, appreciated for their simplicity and effectiveness. Its mathematical formula and graphical representation are depicted in figure 2.3(c). ReLU functions output zero for negative input values, effectively introducing non-linearity while avoiding the vanishing gradient problem encountered by sigmoid functions. This characteristic makes ReLU functions particularly well-suited for training deep neural networks, facilitating faster convergence and alleviating optimization difficulties. Moreover, ReLU functions are computationally efficient, making them an attractive choice for large-scale neural network architectures, where computational resources are often a limiting factor.

Hyperbolic Tangent (tanh), shown in figure 2.3(b), functions represent another common activation function used in neural networks. Similar to sigmoid functions, tanh functions squash input values to a specific range, in this case, between -1 and 1. This wider output range compared to sigmoid functions allows for faster convergence during training. However, like sigmoid functions, tanh functions are susceptible to the vanishing gradient problem, particularly in deep neural networks. Despite this limitation, tanh functions remain widely used in various neural network architectures, especially in scenarios where outputs need to be centered around zero. By leveraging the unique characteristics of each activation function, neural network designers can tailor their models to specific tasks and datasets, optimizing performance and enhancing the overall effectiveness of the network.

2.4 Training Algorithms

Delving into the methodologies employed to optimize the parameters of the neural network, facilitating the learning process. One widely used algorithm is backpropagation, which iteratively adjusts the weights and biases of the network based on the gradient of the loss function with respect to these parameters. Backpropagation involves two main steps: forward propagation, where input data is passed through the network to generate predictions, and backward propagation, where errors are propagated backward through the network to update

the parameters. This iterative process continues until convergence is achieved, minimizing the discrepancy between predicted and actual outputs.

Another commonly employed training algorithm is stochastic gradient descent (SGD), which optimizes the network parameters by computing the gradient of the loss function using only a subset of the training data at each iteration. SGD updates the parameters in the direction that minimizes the loss, making small adjustments to gradually improve performance. By utilizing mini-batches of data, SGD introduces stochasticity into the optimization process, which can help escape local minima and accelerate convergence. Additionally, SGD variants such as Adam and RMSprop incorporate adaptive learning rates and momentum to further enhance convergence speed and stability.

Other advanced optimization algorithms, such as momentum, Adagrad, and Adam, offer enhancements to traditional gradient descent methods by introducing adaptive learning rates, momentum terms, and per-parameter scaling factors. These algorithms aim to accelerate convergence, improve generalization performance, and mitigate issues such as vanishing or exploding gradients. By incorporating these optimization techniques, neural network training can be more efficient and effective, enabling the model to learn complex patterns and relationships within the data more accurately.

2.5 Applications of MLP

The Applications of Multi-Layer Perceptrons (MLPs) span a wide range of domains, showcasing their versatility and effectiveness in solving various real-world problems. In computer vision, MLPs are extensively utilized for tasks such as image classification, object detection, and segmentation. They excel in recognizing patterns and features within images, enabling applications like facial recognition, character recognition, and autonomous driving systems. MLPs have also found significant applications in natural language processing (NLP), where they are employed for tasks like sentiment analysis, language translation, and text generation. Their ability to process sequential data makes them suitable for tasks involving text and speech processing.

Furthermore, MLPs have gained traction in financial modeling and forecasting, where they are used for stock price prediction, risk assessment, and algorithmic trading. Their capability to learn complex nonlinear relationships from data makes them valuable tools for analyzing financial markets and making informed investment decisions. In healthcare, MLPs are utilized for tasks such as disease diagnosis, medical image analysis, and patient outcome

prediction. They play a crucial role in medical imaging techniques like MRI and CT scan analysis, assisting healthcare professionals in diagnosing diseases and planning treatments.

Overall, the applications of MLPs are diverse and widespread, spanning domains such as computer vision, natural language processing, finance, healthcare, and more. Their ability to learn from data and extract meaningful insights makes them indispensable tools for solving complex problems and advancing various fields of study and industry.

2.6 Comparison with other Neural Networks

When comparing Multi-Layer Perceptrons (MLPs) with other neural network architectures, several factors come into play, including model complexity, training efficiency, and performance on various tasks. One common comparison is between MLPs and Convolutional Neural Networks (CNNs), particularly in the context of image processing tasks. CNNs are specifically designed to handle spatial data like images and excel at capturing local patterns and hierarchies of features through their convolutional layers. In contrast, MLPs are more general-purpose and lack the specialized architecture of CNNs, making them less efficient for image-related tasks. However, MLPs can still be effective for tasks where spatial relationships are less critical or when combined with other techniques like dimensionality reduction or feature engineering.

Another comparison is between MLPs and Recurrent Neural Networks (RNNs), which are well-suited for sequential data processing tasks such as natural language processing and time series prediction. RNNs have recurrent connections that allow them to capture temporal dependencies in data, making them effective for tasks requiring memory and context awareness. In contrast, MLPs lack recurrent connections and struggle with sequential data processing, limiting their effectiveness in tasks like sequence generation or language modeling. However, MLPs may outperform RNNs in certain scenarios, such as tabular data classification, where the input features are not sequential.

2.7 Hardware Implementation Challenges

Implementing Multi-Layer Perceptrons (MLPs) in hardware entails addressing various challenges and considerations to ensure efficient deployment. One primary challenge is the management of limited hardware resources, including memory and computational units,

particularly in embedded systems or specialized hardware platforms. MLPs typically demand significant memory for storing weights, activations, and intermediate results, which can strain the constraints of hardware devices. Moreover, the computational demands of MLPs, especially during the forward and backward passes of training, necessitate hardware architectures capable of efficiently handling matrix operations and nonlinear activation functions in real-time.

Another critical consideration is balancing hardware complexity with performance optimization. Designing hardware architectures for MLPs involves striking a delicate balance between flexibility, scalability, resource efficiency, and speed. While complex architectures with extensive parallelism and pipelining can enhance performance, they may consume more resources and require longer development times. Conversely, simpler architectures may offer greater resource efficiency but might compromise on performance or flexibility. Therefore, hardware designers must meticulously assess the application requirements and tailor the hardware architecture accordingly to achieve an optimal balance between performance and resource utilization. Additionally, considerations for power consumption, heat dissipation, and scalability are paramount to ensure energy efficiency, reliability, and adaptability across diverse applications and datasets. By addressing these challenges and considerations, hardware implementations of MLPs can effectively harness the capabilities of neural network models for various real-world applications.

2.8 ZyNet Framework

ZyNet, a versatile python provided hardware framework designed for implementing neural networks on FPGA (Field-Programmable Gate Array) platforms , stands as a testament to the fusion of cutting-edge technology and practical application. With the surge in demand for efficient and high-performance neural network models, ZyNet emerges as a pioneering solution, offering hardware acceleration tailored to meet the computational demands of modern AI algorithms. Built upon FPGA technology, ZyNet harnesses the parallel processing capabilities of hardware to accelerate neural network inference tasks, delivering real-time performance while minimizing power consumption. This introduction encapsulates the essence of ZyNet's innovation, setting the stage for a comprehensive exploration of its architecture, capabilities, and potential impact on the field of artificial intelligence and embedded systems.

2.9 Conversion Process

Within the ZyNet framework, developers benefit from a collection of predefined .tcl (Tool Command Language) scripts encapsulated within the library, streamlining the process of preparing hardware projects using Vivado HLS. These scripts serve as templates or blueprints, automating various aspects of project configuration and generation by modifying a single .tcl file. The user's input, typically pertaining to the desired architecture of the neural network model, is encapsulated within this .tcl file, specifying parameters such as the number of layers and neurons in each layer. Leveraging the flexibility and extensibility of .tcl scripting, developers can easily customize and adapt these templates to accommodate specific project requirements, without the need for extensive manual intervention.

By simply adjusting the parameters within the designated .tcl file, developers can seamlessly tailor the hardware project to their desired neural network architecture, abstracting away the complexities of hardware design and synthesis. This streamlined workflow significantly reduces the time and effort required to instantiate and configure hardware projects, allowing developers to focus on refining the architecture and optimizing performance. Additionally, the standardized approach facilitated by the ZyNet library promotes consistency and repeatability across projects, enhancing collaboration and knowledge sharing within development teams. Overall, the integration of predefined .tcl scripts within ZyNet simplifies the process of hardware project preparation using Vivado HLS, empowering developers to efficiently harness the power of hardware acceleration for neural network applications.

2.10 Other Utilities

The IP generation process in ZyNet involves referencing both the predefined .tcl file available in the library and the .tcl file generated for the specified MLP model. By leveraging these files, ZyNet generates IP blocks tailored to the neural network architecture specified in the model. The predefined .tcl file serves as a template, providing the necessary configurations and settings for the IP generation process. Meanwhile, the model-specific .tcl file contains the detailed specifications of the MLP model, including the number of layers, neurons in each layer, and other relevant parameters. ZyNet utilizes this information to customize the IP blocks according to the specific requirements of the neural network, ensuring compatibility and optimal performance.

Similarly, the block generation process in ZyNet relies on the predefined .tcl file and the model-specific .tcl file to generate customized hardware blocks for the specified MLP model. By combining information from these files, ZyNet creates block designs tailored to the neural network architecture and the target hardware platform. The predefined .tcl file serves as a foundation, providing the initial configuration and settings for the block generation process. Meanwhile, the model-specific .tcl file contains the detailed specifications of the MLP model, enabling ZyNet to customize the block designs according to the specific requirements of the neural network. This approach ensures that the generated hardware blocks are optimized for performance and compatibility with the target hardware environment, facilitating seamless integration and deployment of hardware-accelerated neural network applications.

2.11 Early Works

Data scientists proposed an FPGA-based accelerator optimized for Convolutional Neural Networks (CNNs), emphasizing data optimization techniques to enhance performance. By leveraging FPGA hardware, the proposed accelerator aimed to expedite the computational tasks inherent in CNNs, particularly in the context of real-time applications. Through careful design and implementation, Choi and Kim demonstrated the efficacy of FPGA-based acceleration [15] in improving inference speed and energy efficiency for CNNs. However, the study primarily focused on the acceleration of CNNs and did not extensively explore the scalability or adaptability of the proposed accelerator to other neural network architectures or tasks.

Also, there is introduction of an FPGA-based framework optimized for handwritten digit recognition [16], showcasing the potential of FPGA acceleration in deep learning tasks. Their framework aimed to use the parallel processing capabilities of FPGAs to achieve real-time inference for handwritten digit recognition applications. By exploiting the inherent parallelism in Convolutional Neural Networks (CNNs), the author have demonstrated significant improvements in inference speed and efficiency compared to software-based implementations. However, the study primarily focused on a specific application domain (handwritten digit recognition) and did not explore the generalizability of the framework to other neural network architectures or tasks.

The implementation of an FPGA-based accelerator for deep neural networks, highlighted issues related to resource constraints and scalability. It aimed to address the computational demands of deep neural networks by offloading computations to FPGA hardware. While the FPGA-based accelerator showed promising results in terms of inference

speed and energy efficiency, Tsai et al. [17] noted challenges associated with resource utilization and scalability, particularly when deploying complex neural network models on FPGA platforms. The study underscored the importance of efficient resource management and hardware design optimization in FPGA-based neural network acceleration.

Xiao et al. [18] addressed similar challenges in their study on FPGA implementation of CNNs for handwritten digit recognition, noting limitations in hardware resources and design complexity. It focused on optimizing the hardware architecture of CNNs to maximize resource utilization and inference speed on FPGA platforms. By employing techniques such as parallel processing and hardware/software co-design, they demonstrated improvements in both performance and efficiency compared to software-based implementations. However, the study primarily focused on a specific application domain (handwritten digit recognition) and did not explore the adaptability of the proposed architecture to other neural network tasks or domains.

Single-Stage Convolutional Neural Network (SS-CNN) implemented on an FPGA for handwritten digit recognition, illustrated the trade-offs between accuracy and hardware efficiency. The study aimed to design a hardware-efficient CNN architecture suitable for real-time inference on resource-constrained FPGA platforms. By simplifying the network architecture and reducing computational complexity, Si et al. [19] achieved competitive performance with minimal hardware resources. However, the study primarily focused on optimizing the CNN architecture for a specific task (handwritten digit recognition) and did not explore the generalizability of the proposed SS-CNN architecture to other applications or domains.

In contrast, recent works focus on FPGA acceleration of Multi-Layer Perceptron (MLP) models, showcasing the versatility of FPGA-based implementations across different neural network architectures. Their study aimed to utilize FPGA hardware to accelerate MLP computations for real-time digit recognition applications. By optimizing the hardware architecture and utilizing parallel processing techniques, Westby et al. [20] demonstrated significant improvements in inference speed and efficiency compared to software-based implementations. The study highlighted the potential of FPGA acceleration for MLP models and its applicability to various neural network tasks beyond convolutional architectures.

Chapter

3

Methodology of Proposed Work

This presents a systematic approach to implement a Multilayer Perceptron (MLP) model in hardware using fixed-point arithmetic and the ZyNet framework. Through Python analysis, the optimal fixed-point representation for model computations is determined, and custom classes are developed to handle fixed-point data conversion. The MLP model is then trained using this representation, and the weights are extracted for hardware deployment. Leveraging ZyNet, the trained model is converted into hardware description language (HDL), generating IP cores and block diagrams. FPGA deployment validate the effectiveness of the proposed method, promising efficient hardware acceleration for MLP-based applications.

3.1 Dataset Description

In this proposed work, MNIST [21] dataset, a fundamental resource in machine learning and computer vision introduced by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges in 1998 was used. With its meticulously curated collection of 70,000 hand-written digits, ranging from zero to nine, the MNIST dataset serves as a benchmark for digit recognition tasks. Comprising 60,000 training images and 10,000 test images, each measuring 28x28 pixels, as

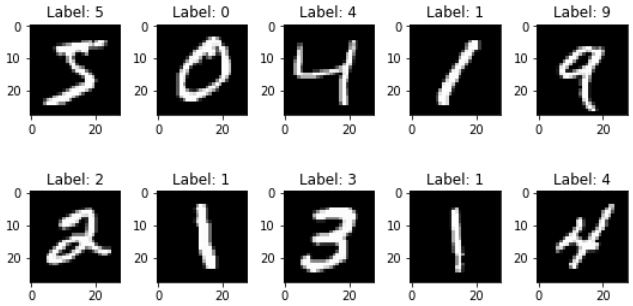


Figure 3.1: Sample images from MNIST dataset

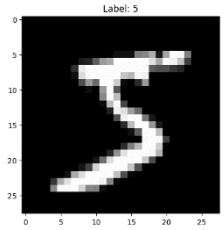


Figure 3.2: 28x28 grayscale image from MNIST

depicted in figure 3.1, the dataset facilitates comprehensive model evaluation. Its balanced distribution ensures thorough assessment of model generalization, making it invaluable for algorithm benchmarking and comparison. The MNIST dataset's simplicity and clarity make it ideal for both novice and seasoned researchers, providing a foundational platform for exploring machine learning algorithms and techniques in image classification and pattern recognition. Sample images are shown in figure 3.2.

3.2 Fixed-Point Analysis

In the initial phase of the proposed work, an exhaustive examination was conducted of fixed-point arithmetic, known as Fixed Point Analysis. This thorough investigation aimed to unravel the intricate relationship between various parameters, particularly the number of fractional bits within an 8-bit framework, and their influence on key performance metrics essential for evaluating the model. The mean squared loss (MSE), accuracy, and training time are meticulously scrutinized of this MLP model, systematically varying the number of fractional bits to discern its impact on model behavior and performance. Through this process, it has been aimed to make informed decisions regarding the optimal configuration for fixed-point representation.

To visually illustrate the findings and provide tangible insights, a series of informative plots generated. These plots offer compelling visual representations of the relationships between the number of fractional bits and key performance indicators. For instance, the plot depicting Training Time versus the Number of Fractional Bits provides insights into the temporal implications of different fixed-point configurations on the model's training process. Similarly, the Accuracy versus Number of Fractional Bits plot showcases the trade-offs between computational accuracy and the granularity of fixed-point representation. Additionally, the plot displaying Mean Squared Error (MSE) against the Number of Fractional Bits offers valuable insights into the impact of fixed-point precision on model convergence and error minimization. Together, these plots offer a comprehensive understanding of the multifaceted implications of fixed-point arithmetic, guiding subsequent stages of the methodology.

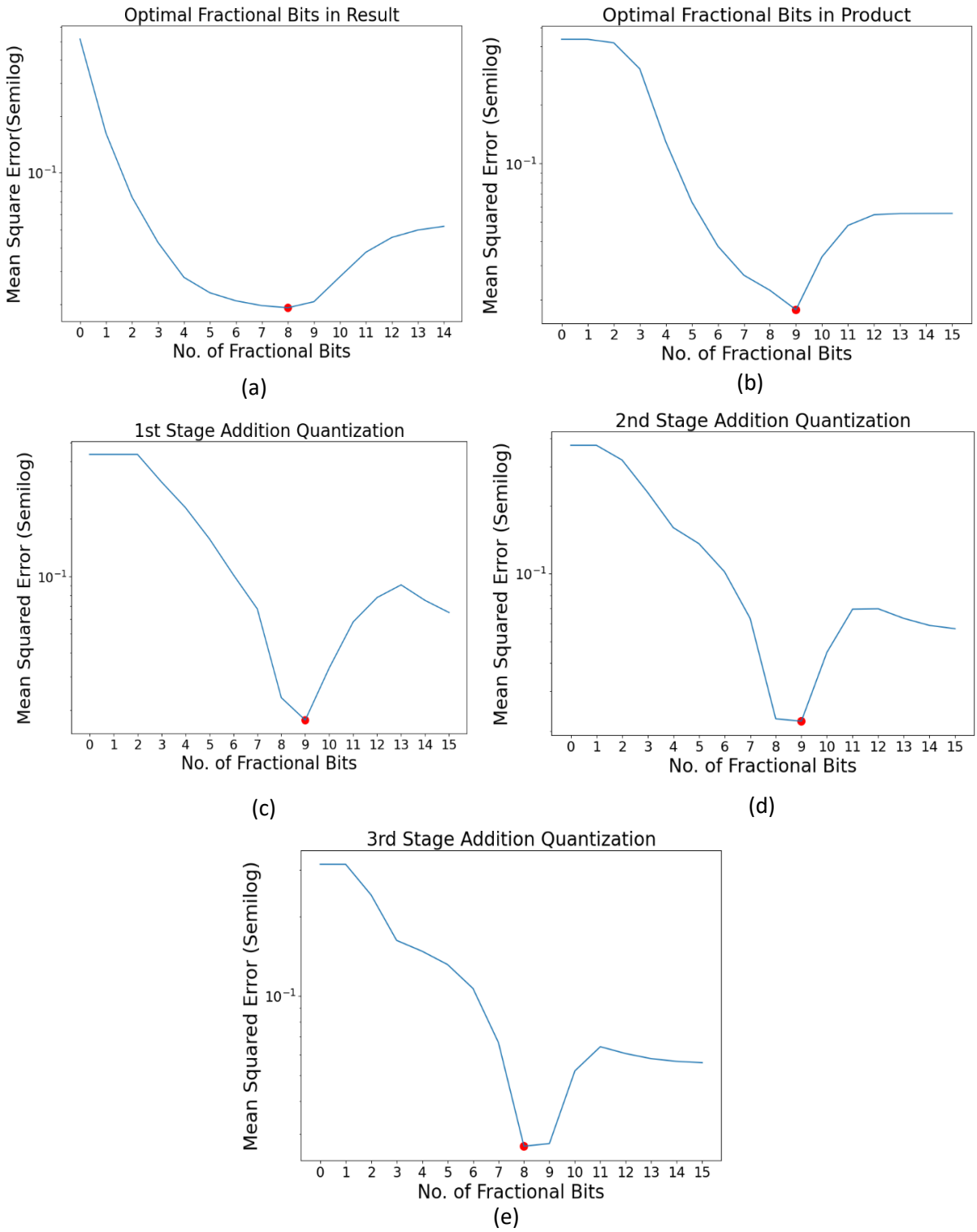


Figure 3.3: Different Stages of Quantization: (a) result stage, (b) product stage, (c) 1st stage addition, (d) 2nd stage addition, (e) 3rd stage addition

3.3 Training the Quantized Model

The exploration of various bit configurations uncovered that the combination of 8 bits and 8 fractional bits consistently yielded the lowest error rates across the studied metrics. This discovery prompts the decision to adopt this particular configuration for quantizing both the input images and kernel values, as well as subsequent feature maps. To elaborate, quantization involves mapping continuous numerical values to discrete representations, thereby reducing the computational complexity and memory footprint of neural network operations. By aligning the precision of the data representations with the identified optimal configuration, aimed to maintain model accuracy while capitalizing on the efficiency gains afforded by reduced precision computation.

The neural network architecture chosen for this endeavor is a multilayer perceptron (MLP) comprising four layers, each serving a distinct role in the information processing pipeline. Beginning with an input layer housing 784 neurons, corresponding to the flattened representation of 28x28 pixel images from the MNIST dataset, the network progresses through two hidden layers featuring 30 and 20 neurons, respectively, before culminating in an output layer with 10 neurons, representing the ten possible digit classes. This architecture design reflects a balance between model complexity and computational efficiency, enabling effective learning while mitigating the risk of overfitting on the training data.

During the training process, a suite of hyperparameters is carefully curated to guide the optimization dynamics and foster model convergence. Stochastic gradient descent (SGD) emerges as the optimizer of choice, offering robustness and scalability for training MLPs. Complementing this optimization strategy is a fixed learning rate of 0.01, which governs the magnitude of weight updates during gradient descent, balancing the need for rapid convergence with stability. To adaptively modulate the learning rate in response to the model's performance on validation data, a scheduler mechanism incorporated known as ReduceOnPlateau. This dynamic adjustment mechanism systematically decreases the learning rate when the monitored metric, such as validation loss, stagnates or worsens over successive epochs, thus facilitating finer-grained control over the optimization process. Additionally, to safeguard against potential setbacks and ensure continual progress, a checkpointing mechanism is implemented to save the model's state at key intervals, specifically whenever the validation loss improves. This proactive approach preserves the best-performing model parameters, enabling seamless recovery and continued advancement throughout the training regimen.

3.4 Extraction of Weights and Biases

A critical step taken towards hardware implementation by transforming the learned parameters of the trained MLP model into a format suitable for deployment on FPGA hardware. This process unfolds in several stages, beginning with the extraction of weights and biases from the trained model. These parameters, encapsulating the learned relationships between network layers and facilitating accurate prediction, are pivotal for the operational efficacy of the deployed hardware model.

Upon extraction, the weights undergo quantization to align with the chosen precision configuration of 8 bits and 8 fractional bits, ensuring compatibility with the targeted hardware architecture. This quantization step involves discretizing the continuous weight values into fixed-point representations, preserving the essential characteristics of the original weights while adhering to the specified bit precision constraints. Subsequently, the quantized weights are transformed into binary format, a prerequisite for their utilization within the hardware environment. This conversion process entails encoding the discrete weight values as sequences of binary digits, facilitating efficient storage and computation within FPGA hardware. To

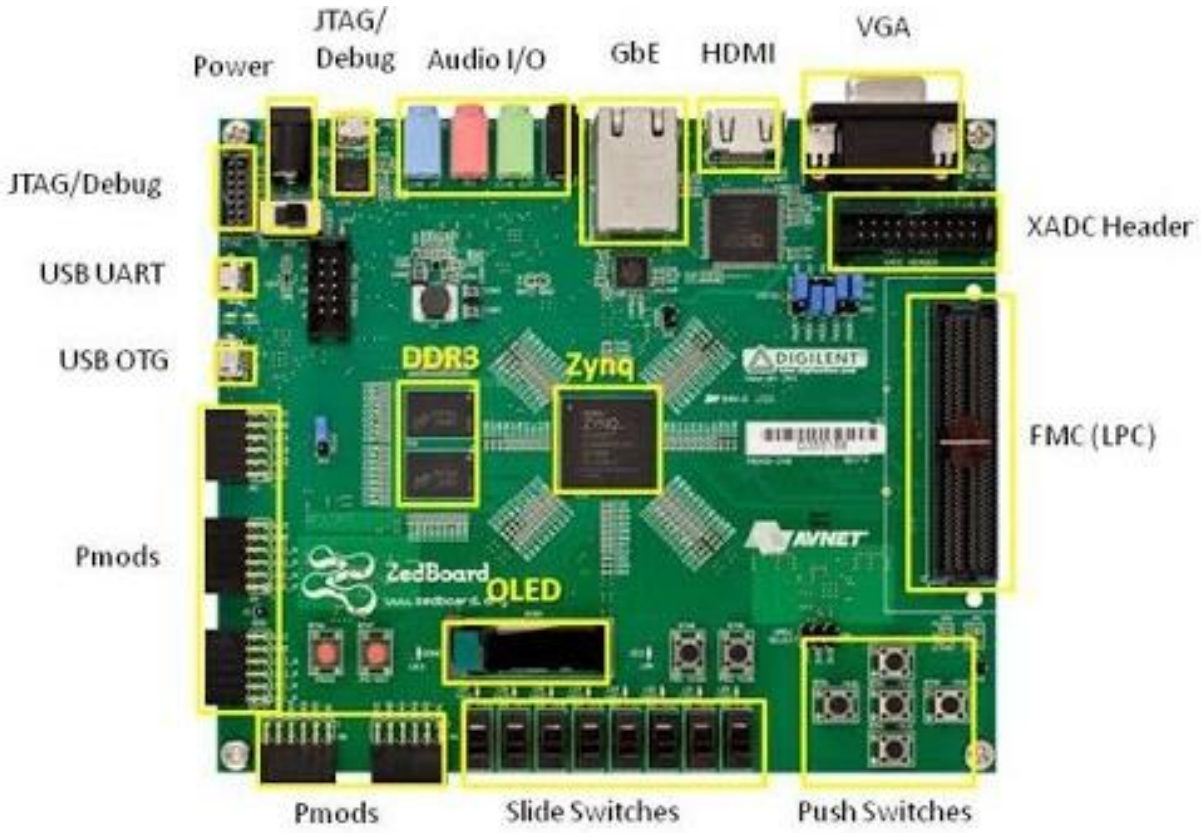


Figure 3.4: Zedboard Zynq Evaluation and Development Kit (xc7z020clg484-1)

facilitate seamless integration with the FPGA platform, the quantized and binary-encoded weights are then serialized into Memory Initialization Files (MIF). These files serve as repositories for the structured storage of memory contents, organizing the weights into a format conducive to direct interfacing with the FPGA memory subsystem. By encapsulating the weight data within MIF files, the integration process streamlined, enabling straightforward access and utilization of the trained model parameters within the FPGA hardware environment. This meticulous preparation of the weight data sets the stage for the subsequent stages of hardware synthesis and deployment, laying a robust foundation for the realization of efficient and scalable hardware-accelerated inference capabilities.

3.5 Build the MLP in Verilog HDL using ZyNet

The process of constructing the hardware implementation of the MLP model unfolds with the utilization of ZyNet, a specialized framework tailored for this purpose. Initially, the creation of the hardware project is initiated through a dedicated method provided by the ZyNet library. This method accepts essential parameters such as the number of layers, the number of neurons in each layer, the name of the evaluation kit utilized, and the designated project name. Notably, the evaluation board employed in this endeavor is the Zedboard Zynq xc7z020clg484-1, illustrated in figure 3.4 a versatile platform renowned for its suitability in FPGA-based development tasks. Leveraging the capabilities of Vivado HLS command-line interface, ZyNet orchestrates the creation of the project, automating the requisite tasks while ensuring alignment with the specified architectural parameters.

Upon successful project creation, ZyNet generates a .tcl (Tool Command Language) file, encapsulating the project configuration and specifications. This .tcl file serves as a crucial artifact for subsequent stages of the workflow, facilitating seamless integration and coordination across diverse toolchains and methodologies. Subsequently, the framework proceeds to the generation of Intellectual Property (IP) for the project, leveraging the capabilities of Vivado HLS to package the project as a deployable device. This IP encapsulates the hardware implementation of the MLP model, embodying the intricate computational and connectivity logic required for efficient inference processing.

With the IP generation completed, ZyNet transitions to the creation of the block design, a pivotal step in tailoring the hardware implementation to the specifications of the target evaluation board. By adhering to the board's unique requirements and constraints, ZyNet ensures the compatibility and optimality of the hardware design, setting the stage for seamless

integration and deployment onto the selected FPGA platform. Through this meticulously orchestrated process, ZyNet empowers practitioners to transform abstract MLP models into efficient and scalable hardware implementations, facilitating the realization of high-performance neural network inference capabilities on FPGA-based embedded systems.

The block diagram of the MLP is depicted in figure 3.5.

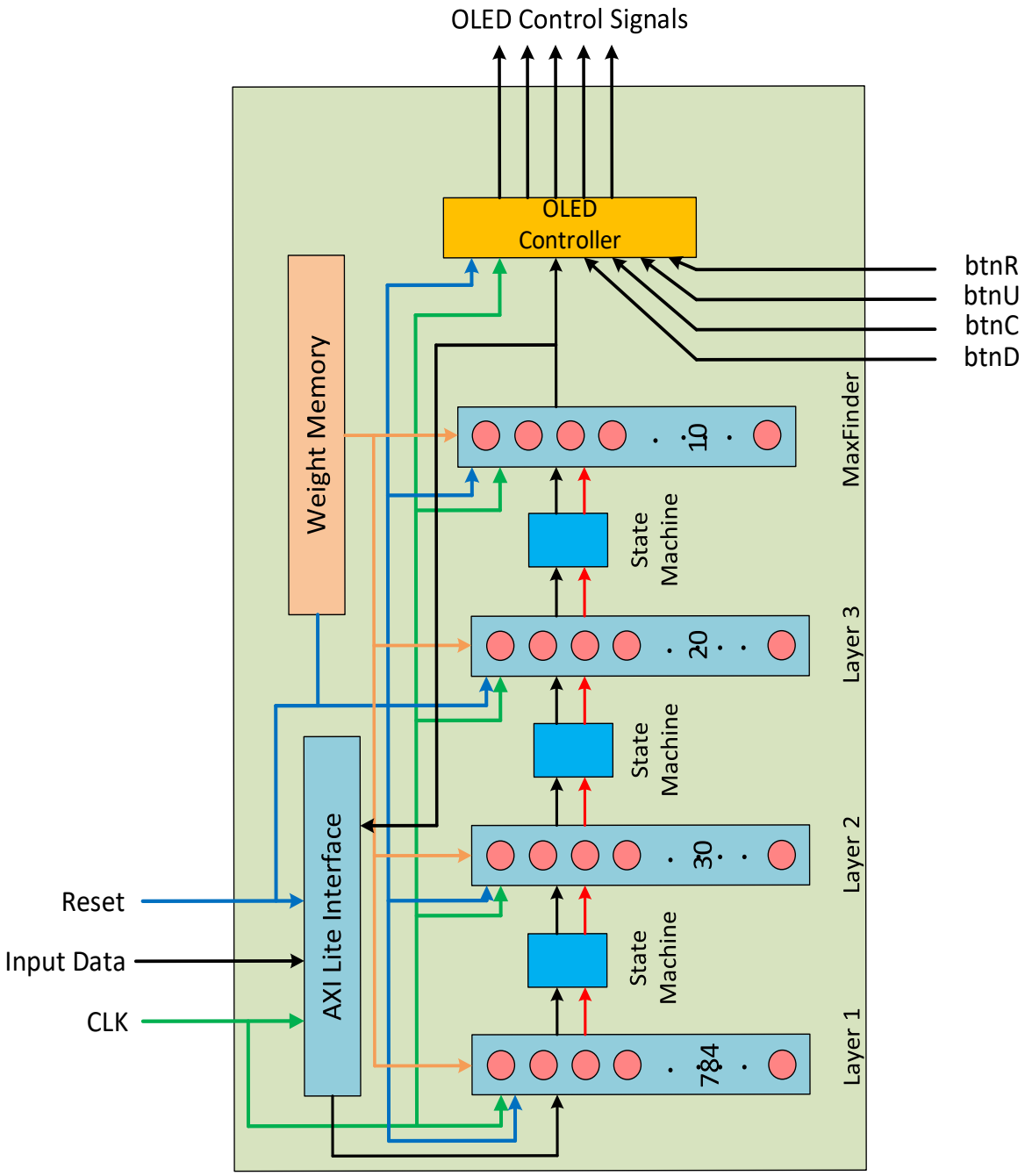


Figure 3.5: Block Diagram for proposed MLP designed using ZyNet

3.6 Layer Module in HDL

The Verilog code provided defines a module named "Layer_1," which represents the first layer of a neural network. This module is parameterized to accommodate various configurations, including the number of neurons (NN), the number of weights per neuron (numWeight), the data width for input/output signals (dataWidth), the layer number (layerNum), the size of the sigmoid function (sigmoidSize), the width of weight integers (weightIntWidth), and the activation function type (actType).

Within the module, there are instances of a submodule called "neuron," each representing an individual neuron within the layer. These neuron instances are instantiated sequentially, with each neuron receiving input signals and parameters from the parent module and producing output signals. The input signals to the module include clock (clk), reset (rst), signals indicating the validity of weight and bias values (weightValid, biasValid), weight and bias values themselves (weightValue, biasValue), configuration parameters such as layer number and neuron number (config_layer_num, config_neuron_num), input data validity (x_valid), and input data (x_in).

The output signals include signals indicating the validity of output data (o_valid) and the output data itself (x_out). Each neuron instance operates independently, processing input data and applying weights and biases to produce output data. The weights and biases used by each neuron are specified by separate MIF (Memory Initialization File) files, enabling flexibility and configurability in the neural network architecture.

Overall, the "Layer_1" module orchestrates the operation of multiple neurons, facilitating the transformation of input data through a layer of the neural network, paving the way for subsequent layers to further process and analyze the data.

3.7 Neuron Module in HDL

At its core, the `neuron` module is structured to handle various aspects of neuron computation. It begins by defining its interface ports, which facilitate communication with other modules in the system. These ports include inputs for clock signals (`clk`), reset signals (`rst`), input data (`myinput`), and various control signals (`myinputValid`, `weightValid`, `biasValid`, etc.), as well as an output port for the computed result (`out`). By providing these ports, the module establishes a means for exchanging information with the broader neural network architecture.

One key aspect of the `neuron` module is its internal state management, facilitated by a series of registers and wires. These elements orchestrate the flow of data within the module, controlling operations such as weight memory access, multiplication, addition with bias, and activation function application. For instance, the module maintains registers such as `wen` (write enable), `w_addr` (write address), and `w_in` (input data to weight memory) to manage interactions with the weight memory component. Additionally, it employs registers like `mul` (multiplication result), `sum` (sum of multiplication result and bias), and `bias` (bias value) to perform the core computations required by the neuron.

A critical aspect of neural network implementation is the handling of activation functions, which introduce non-linearity into the network's behavior. In this module, the type of activation function applied is configurable via the `actType` parameter. Depending on this parameter, the module instantiates either a sigmoid function (`Sig_ROM`) or a Rectified Linear Unit (ReLU) function (`ReLU`). These functions are essential for introducing non-linear transformations to the neuron's output, enabling the network to learn complex patterns and relationships within the input data.

Furthermore, the `neuron` module incorporates features for flexibility and configurability. Parameters such as `layerNo`, `neuronNo`, and `numWeight` allow the module to adapt to different positions within the neural network and accommodate varying network architectures. Additionally, support for loading bias values from a file (`biasFile`) provides a mechanism for initializing the neuron's behavior with pre-trained parameters, enhancing its usability and versatility.

3.8 ReLU Module in HDL

The `ReLU` module, short for Rectified Linear Unit, embodies a simple yet effective activation function commonly used in neural networks to introduce non-linearity. This Verilog module is designed to operate within a digital hardware environment and is parameterized to accommodate different data widths (`dataWidth`) and integer bit widths (`weightIntWidth`). At its core, the `ReLU` module evaluates the input signal `x` and applies the ReLU function to produce the output signal `out`. The ReLU function is defined as follows: for any input value `x`, if `x` is greater than or equal to zero, the output is `x` itself; otherwise, if `x` is negative, the output is zero.

To implement this behavior, the module contains an `always` block sensitive to the rising edge of the clock signal (`clk`). Within this block, the input signal `x` is examined using

conditional logic. If `x` is greater than or equal to zero (indicating the positive range), the module checks whether the most significant bits of `x` represent an overflow condition into the sign bit of the integer part. If an overflow is detected, indicating that the input value is too large to represent, the output is saturated to the maximum positive value to prevent overflow issues. Otherwise, the output simply retains the original value of `x`, truncated to the appropriate data width. On the other hand, if `x` is negative, the output is set to zero, adhering to the behavior of the ReLU activation function.

3.9 Sigmoid Module in HDL

The module represents a ROM (Read-Only Memory) designed to implement a lookup table for the sigmoid function. Sigmoid functions are commonly used as activation functions in neural networks due to their smooth, non-linear behavior. This Verilog module is parameterized to accommodate different input widths (`inWidth`) and data widths (`dataWidth`), offering flexibility in its application. The module consists of two main components: a memory array (`mem`) and a computation block within an `always` block sensitive to the rising edge of the clock signal (`clk`). Additionally, there is an initialization block that reads the content of the ROM from a memory initialization file (`sigContent.mif`) and stores it in the memory array `mem`.

During operation, the input signal `x`, representing the input to the sigmoid function, is received by the module. The `always` block calculates the memory address `y` based on the value of `x`. If `x` is positive or zero, `y` is calculated as `x + 2^(inWidth-1)`, effectively shifting the range of `x` to be positive, since the sigmoid function is defined over the range [0, 1]. If `x` is negative, `y` is calculated as `x - 2^(inWidth-1)` to ensure that negative values are also properly represented within the positive range. The computed memory address `y` is then used to index into the memory array `mem`, retrieving the corresponding value stored in the ROM. This value represents the output of the sigmoid function for the given input `x`.

3.10 Weight Memory Module in HDL

The module consists of a memory array (`mem`) where weight values are stored. This array is parameterized based on the number of weights (`numWeight`) and the data width (`dataWidth`) of each weight value. The memory array is declared as a register array to retain its values across clock cycles.

The module's input ports include the clock signal (`clk`) for synchronization, control signals for write enable (`wen`) and read enable (`ren`), address inputs for write (`wadd`) and read (`radd`) operations, and data input (`win`) for writing weight values into memory. The output port `wout` provides the retrieved weight value during read operations.
Conditional compilation directives (`ifdef`) are used to handle two different modes of operation:

1. **Pretrained Mode**: In this mode, weights are loaded into the memory array during initialization from a memory initialization file specified by the `weightFile` parameter. The `initial` block reads the content of the file and initializes the memory array accordingly.

2. **Normal Operation Mode**: If the `pretrained` directive is not defined, the module operates in normal mode. In this mode, weights are written into the memory array (`mem`) during execution based on the write enable signal (`wen`).

The `always` block sensitive to the rising edge of the clock (`posedge clk`) handles the writing operation. When `wen` is asserted, indicating a write operation, the input weight value (`win`) is written into the memory array at the specified address (`wadd`). Regardless of the mode of operation, the module always responds to read requests. The `always` block for read operations triggers on the rising edge of the clock (`posedge clk`). When `ren` is asserted, indicating a read operation, the module retrieves the weight value stored at the specified address (`radd`) from the memory array and outputs it through the `wout` port.

3.11 MaxFinder in HDL

At its core, the module consists of an always block triggered by the positive edge of the clock (`i_clk`). Within this block, the module manages the state of the input data stream, tracks the maximum value encountered, and determines the index of the maximum value. Upon receiving a valid input (`i_valid`), the module initializes by storing the first input data slice into the `maxValue` register, setting the `counter` to 1 to track the number of input data slices processed, storing the entire input data buffer (`inDataBuffer`), and resetting the output data (`o_data`) to 0. As the input data stream continues (`i_valid` remains high), the module iterates through each subsequent input data slice. It compares each slice with the current maximum value

(`maxValue`) stored in the `maxValue` register. If a new maximum value is found, the `maxValue` register is updated, and the index of the new maximum value (`counter`) is stored in the `o_data` register.

The module increments the `counter` with each new input data slice to keep track of the number of slices processed. When the number of processed slices equals the specified number of inputs (`numInput`), the module signals that the output data (`o_data`) is valid by setting `o_data_valid` to 1. Throughout this process, the module continually updates the maximum value encountered (`maxValue`) and its corresponding index (`o_data`) until all input data slices have been processed. Once the input data stream ends, the module outputs the index of the maximum value and indicates the validity of the output data, thus completing its operation.

3.12 AXI Lite Wrapper in HDL

The `axi_lite_wrapper` module serves as a crucial component for facilitating communication between a master device and a slave device using the Advanced Extensible Interface (AXI) Lite protocol. Through its defined interface signals, it establishes connections for transmitting data, control signals, and addressing information between the master and the slave. These interface signals include clock (`S_AXI_ACLK`), reset (`S_AXI_ARESETN`), write address (`S_AXI_AWADDR`), write data (`S_AXI_WDATA`), write strobes (`S_AXI_WSTRB`), write valid (`S_AXI_WVALID`), read address (`S_AXI_ARADDR`), and read valid (`S_AXI_ARVALID`), among others, catering to both write and read transactions.

Within the module, a series of registers are declared to manage the flow of data and control signals. These registers, such as `axi_awaddr`, `axi_awready`, `axi_wready`, `axi_bresp`, `axi_bvalid`, `axi_araddr`, `axi_arready`, and `axi_rdata`, play pivotal roles in synchronizing and processing the incoming and outgoing signals. Additionally, user-specific registers are included to store data relevant to the specific application or functionality being implemented. Clock and reset handling mechanisms are integrated within the module to ensure proper synchronization and initialization. Synchronization with the global clock signal (`S_AXI_ACLK`) is essential for coordinating the timing of operations, while the system reset (`S_AXI_ARESETN`) ensures correct initialization and recovery from reset conditions, guaranteeing the reliability of the module's operation.

The module implements logic to handle both write and read transactions effectively. For write transactions, it latches the write address and data when both write address valid and

write data valid signals are asserted. Similarly, for read transactions, it latches the read address and provides the requested data when read valid and read ready signals are both asserted, ensuring seamless communication between the master and the slave.

Moreover, the module incorporates memory-mapped register handling logic, which decodes the address provided by the master and controls the read and write operations to different registers based on the address. This functionality enables efficient management of data and control information, enhancing the overall performance and flexibility of the module. Furthermore, the module includes user-specific logic to accommodate custom functionalities or operations required by the application. This allows for the integration of specialized features or processing capabilities tailored to the specific needs of the system or application utilizing the module.

3.13 OLED Control

The module, named `top`, receives several input signals representing clock (`clk`) and various button inputs (`btnR`, `btnC`, `btnD`, `btnU`). Additionally, it has output signals for controlling the OLED display (`oled_sdin`, `oled_sclk`, `oled_dc`, `oled_res`, `oled_vbat`, `oled_vdd`) and an LED output (`led`). It also takes an input `detected_digit` which is a 32-bit vector representing the detected digit. Inside the module, there's a state machine implemented using `reg` variables to manage the different states of operation. Each state corresponds to a different functionality, such as initialization, active display, writing to the display memory, etc. State transitions are triggered by button inputs and other conditions.

The module includes parameters and local parameters to define various states and configurations, making it easy to adjust its behavior and functionality. There are also several strings defined (`str1`, `str2`, `str3`, `str4`) representing the text to be displayed on the OLED screen. The value of `detected_digit` determines the content of `str3` through an `always` block that sets `str3` based on the detected digit value.

Button inputs are debounced using separate debouncer modules (`get_dBtnC`, `get_dBtnU`, `get_dBtnD`, `get_rst`) to ensure reliable operation even with noisy signals. The OLED display control signals (`update_start`, `disp_on_start`, `disp_off_start`, `toggle_disp_start`, `write_start`) are managed based on the state machine's current state and button inputs. There's also logic to handle writing characters to the OLED display memory based on the selected string and position.

Finally, the state machine transitions between different states based on button inputs (`rst`, `dBtnC`, `dBtnU`, `dBtnD`) and other conditions, controlling the overall operation of

the module. At the end of the Verilog module, it's inferred that there's an IP core for the OLED controller instantiated (`OLEDCtrl`) to handle the low-level interfacing with the OLED display hardware. This module effectively integrates the functionality of detecting digits and displaying them on an OLED screen, controlled by button inputs and managed by a state machine.

3.14 MLP module in HDL

The module is named `MLP` and has several input and output ports, including clock and reset signals for the AXI interface, AXI Stream interface for input data (`axis_in_data`, `axis_in_data_valid`, `axis_in_data_ready`), AXI Lite interface for control and configuration (`s_axi_awaddr`, `s_axi_awprot`, `s_axi_awvalid`, etc.), an interrupt output (`intr`), button inputs (`btnR`, `btnC`, `btnD`, `btnU`), and various signals for controlling the OLED display (`oled_sdin`, `oled_sclk`, etc.). The module consists of multiple instances of layers (`Layer_1`, `Layer_2`, `Layer_3`) representing different layers of a neural network. Each layer performs computations on the input data and produces output data. The output of one layer serves as the input to the next layer. There are also state machines (`state_1`, `state_2`, `state_3`) for data pipelining, which manage the flow of data between layers. These state machines ensure that data is processed sequentially through the layers. A `maxFinder` module is used to determine the maximum output value from the neural network layers.

The `axi_lite_wrapper` module handles the AXI Lite interface, allowing the external controller to configure the neural network accelerator. The `oled_top` module integrates control of the OLED display, allowing it to display information such as detected digits. Overall, this Verilog module combines neural network processing with AXI interface and OLED display control, making it suitable for applications requiring real-time neural network inference with display output.

After the synthesis of above is completed, power report (figure 3.6), utilization report (figure 3.7), timing report (figure 3.8) and schematic diagram (figure 3.9) are generated.

3.15 IP Generation and Block Diagram

The culmination of the MLP project is the creation of a highly versatile Intellectual Property (IP) package, christened "mlp_oled,"(figure 3.6) which encapsulates the intricate functionalities

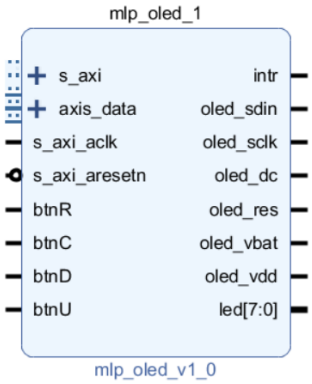


Figure 3.6: IP of MLP project

meticulously crafted within the top module. This IP, meticulously crafted and refined through the project endeavors, stands as a testament to the depth of the exploration into Multilayer Perceptron (MLP) architectures. It represents the culmination of extensive design, development, and optimization efforts aimed at harnessing the potential of MLPs within FPGA environments. With its carefully defined input and output ports, the "mlp_oled" IP package serves as a robust foundation for a myriad of future applications and endeavors. Its creation marks a significant milestone in the journey toward leveraging FPGA-based implementations for neural network inference tasks, paving the way for further innovation and exploration in the realm of hardware-accelerated machine learning.

The block design for integrating the IP in Zedboard Zynq Evaluation and Development kit (xc7z020clg484-1) is created in Vivado HLS by ZyNet framework is depicted in figure 3.7.

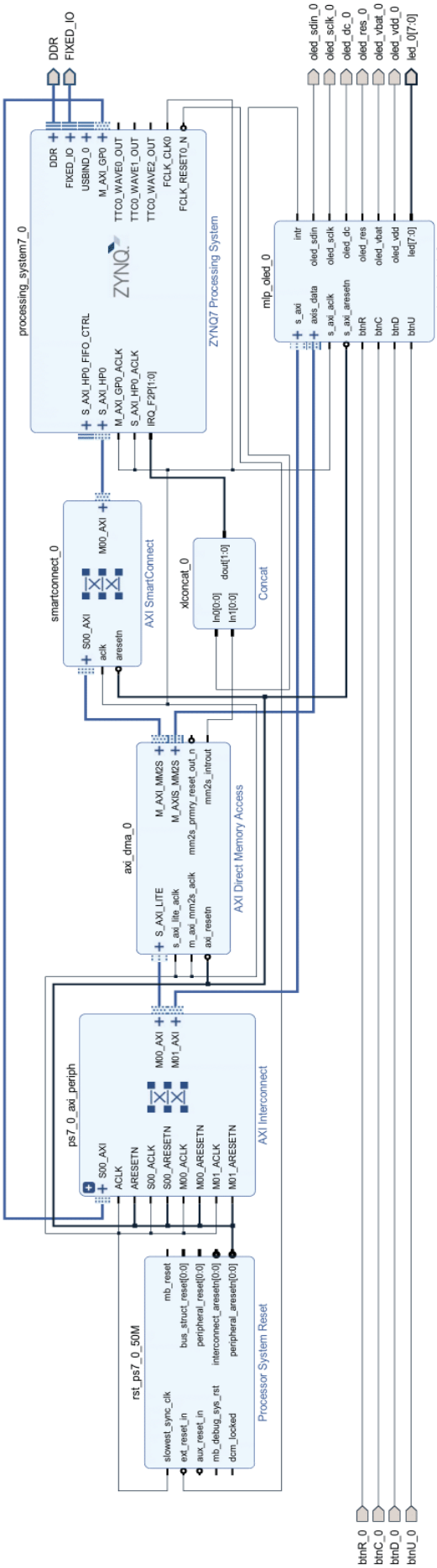


Figure 3.7: Block Design of the MLP project where IP is connected as a peripheral.

3.16 Programming in SDK

After the bitstream is generated, the programming journey delves into a meticulous blend of Python scripting and C programming, each serving a crucial role in the intricate process. Python scripts take the stage first, orchestrating the generation of diverse test data that finds its haven in header files. This data, meticulously crafted, stands poised to fuel the neural network accelerator with a plethora of inputs, ready to unravel its computational prowess. With the test data at the helm, the narrative pivots towards the realm of C programming, where a bespoke program is meticulously crafted to orchestrate the intricate dance between the FPGA board and its environment. This C program wields the power to dispatch the meticulously prepared test data to the FPGA's gates, eagerly awaiting the neural network's verdict.

Yet, in this symphony of data and logic, an intriguing twist emerges: the output, rather than being confined to the cold digital embrace of a UART terminal, finds solace in the warm luminescence of an OLED display. This shift in perspective not only streamlines the process but also elevates the experience, transforming the output into a tangible, visual manifestation that graces the Evaluation kit's OLED screen. As the FPGA is imbued with the intricacies of the neural network design, the programming port becomes the conduit through which the FPGA's transformation is realized, culminating in the launch of the SDK application. With bated breath, the developers witness the culmination of their efforts, as the OLED display becomes the canvas upon which the neural network's insights are vividly painted, a testament to the fusion of hardware and software ingenuity. In this convergence of disciplines, the journey from bitstream to OLED showcase represents not merely a programming task but a symphony of creativity, engineering, and innovation.

3.17 Results

The power report, utilization report, timing report and schematic diagram are depicted in figure 3.8, 3.9, 3.10 and 3.11 respectively. The results of the proposed method showcase its remarkable efficiency and effectiveness when compared to various early works in the field. The method, deployed on the Zedboard Zynq xc7z020clg484-1, achieves an outstanding accuracy of 99% while utilizing only 10,301 LUTs. This exceptional performance underscores the potency and efficacy of this approach in delivering high accuracy with minimal resource utilization.

In contrast, a survey of other seminal works, illustrated in table 3.1, reveals a spectrum of outcomes across different models and FPGA kits. For instance, the LeNet-5 CNN,

Model	FPGA	Accuracy	Hardware Costs (LUTs used)
LeNet-5 CNN [16]	Xilinx Zynq 7Z020-100	90-96%	18,426
DNN [17]	Xilinx Zynq 7Z020-100	94.67%	38,899
CNN [18]	Intel Cyclone10-150	97.57%	12,588
SS-CNN [19]	Intel Cyclone IVE-30	98.8%	98,000
LeNet-5 CNN HLS [15]	Xilinx Zynq 7Z020-100	98.64%	33,585
Proposed	Zynq xc7z020clg484-1	99%	10,301

Table 3.1: Comparison with similar works

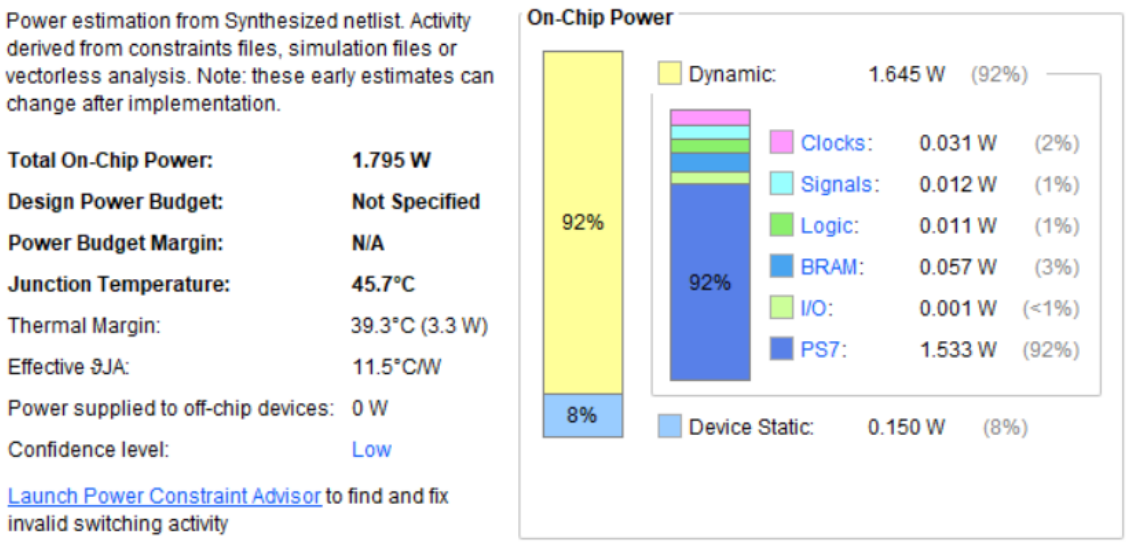


Figure 3.8: Power Report

Resource	Utilization	Available	Utilization %
LUT	10301	53200	19.36
LUTRAM	312	17400	1.79
FF	6660	106400	6.26
BRAM	28.50	140	20.36
IO	18	200	9.00

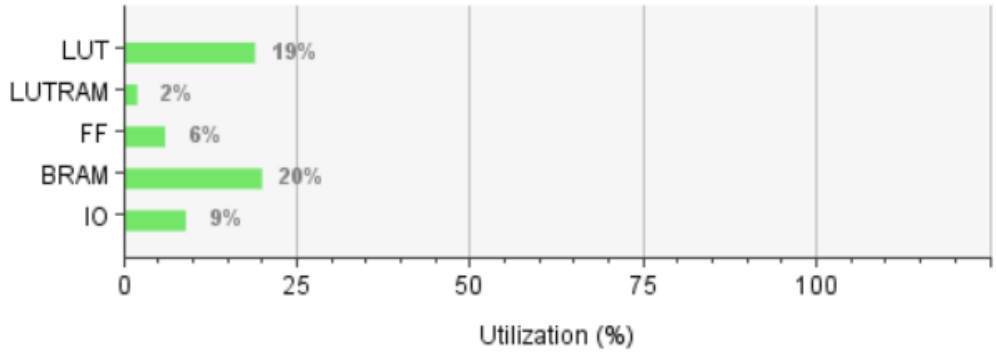


Figure 3.9: Utilization Report

Pad	Max Delay	Max Edge	Max Process Corner	Min Delay	Min Edge	Min Process Corner	Edge Skew
led_0[0]	7.467	Rise	SLOW	5.728	Rise	SLOW	0.000
led_0[0]	3.504	Rise	FAST	2.372	Rise	FAST	0.000
led_0[1]	-∞	Rise/Fall	SLOW	∞	Rise/Fall	SLOW	-
led_0[1]	-∞	Rise/Fall	FAST	∞	Rise/Fall	FAST	-
led_0[2]	-∞	Rise/Fall	SLOW	∞	Rise/Fall	SLOW	-
led_0[2]	-∞	Rise/Fall	FAST	∞	Rise/Fall	FAST	-
led_0[3]	-∞	Rise/Fall	SLOW	∞	Rise/Fall	SLOW	-
led_0[3]	-∞	Rise/Fall	FAST	∞	Rise/Fall	FAST	-
led_0[4]	-∞	Rise/Fall	SLOW	∞	Rise/Fall	SLOW	-
led_0[4]	-∞	Rise/Fall	FAST	∞	Rise/Fall	FAST	-
led_0[5]	-∞	Rise/Fall	SLOW	∞	Rise/Fall	SLOW	-
led_0[5]	-∞	Rise/Fall	FAST	∞	Rise/Fall	FAST	-
led_0[6]	-∞	Rise/Fall	SLOW	∞	Rise/Fall	SLOW	-
led_0[6]	-∞	Rise/Fall	FAST	∞	Rise/Fall	FAST	-
led_0[7]	-∞	Rise/Fall	SLOW	∞	Rise/Fall	SLOW	-
led_0[7]	-∞	Rise/Fall	FAST	∞	Rise/Fall	FAST	-
Worst Case Summary	7.467	Rise	SLOW	5.728	Rise	SLOW	0.000
Worst Case Summary	3.504	Rise	FAST	2.372	Rise	FAST	0.000

Figure 3.10: Timing Report

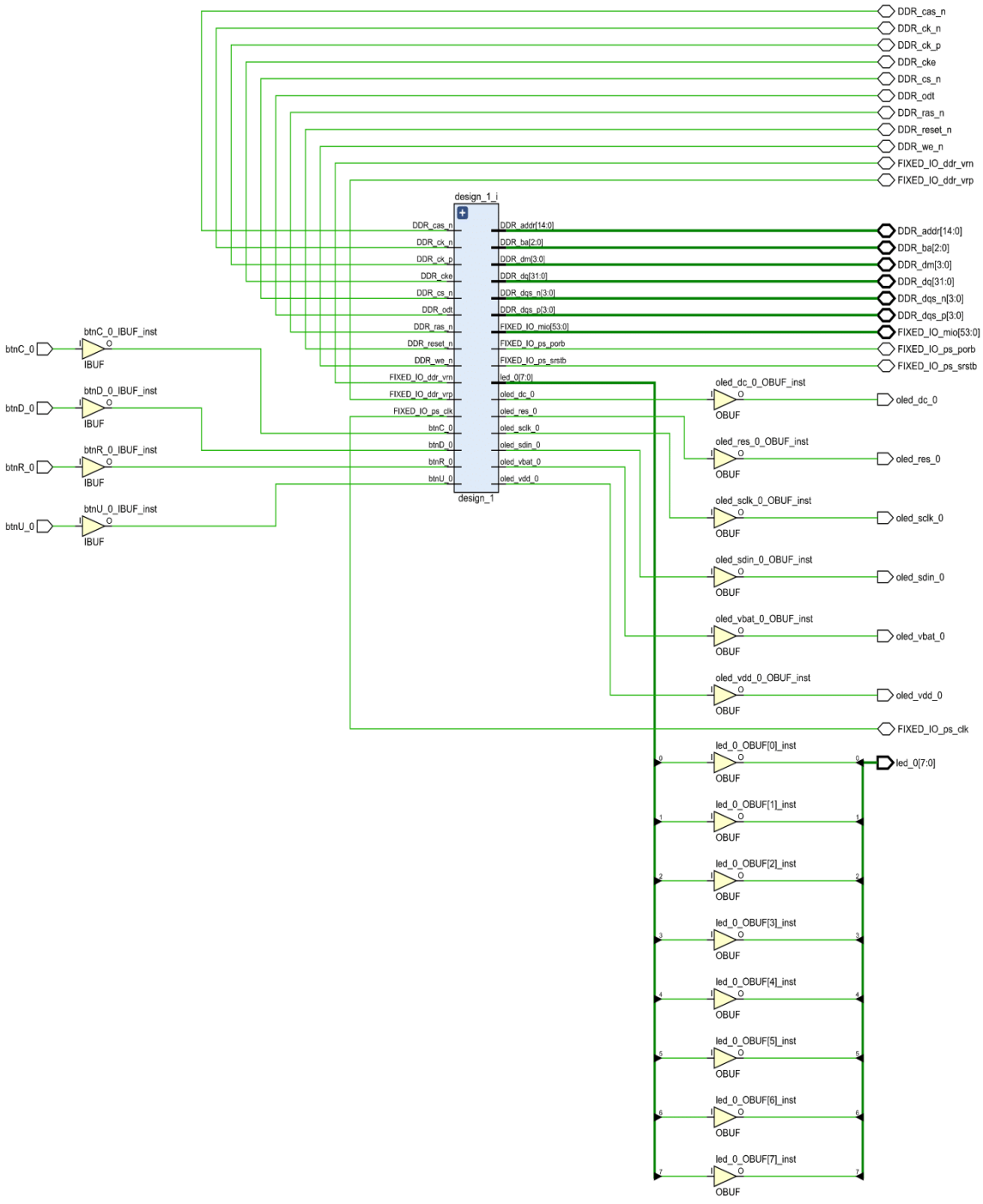


Figure 3.11: Schematic Diagram of Synthesized Design

implemented on the Xilinx ZCU102-100, achieves an accuracy of 98.64% but requires a significantly higher resource utilization of 32,589 LUTs. Similarly, implementations of LeNet-5 CNN and DNN on the Xilinx Zynq 7Z020-100 exhibit accuracies ranging from 90% to 96% and 94.67%, respectively, with resource utilizations of 18,426 LUTs and 38,899 LUTs.

Comparatively, the proposed method not only surpasses these accuracy benchmarks but also excels in resource efficiency. For instance, while achieving a higher accuracy of 99%, the method consumes notably fewer LUTs compared to these early works. This superior resource efficiency is further exemplified when contrasted with implementations on other FPGA kits such as the Intel Cyclone10-150 and Cyclone IVE-30, where accuracies range from 97.57% to 98.8% but resource utilizations soar as high as 98,000 LUTs.

Moreover, the method stands out for its power efficiency, as evidenced by the power report which indicates a remarkably low power usage of only 1.795W. This demonstrates the method's ability to deliver exceptional performance while maintaining energy efficiency, further solidifying its appeal for deployment in power-constrained environments.

Furthermore, to provide a comprehensive understanding of this proposed method, output on the FPGA evaluation kit has been depicted in figure 3.10. These visual representations offer insights into the architectural intricacies and structural composition of the approach, facilitating a deeper appreciation of its underlying mechanisms and optimizations.

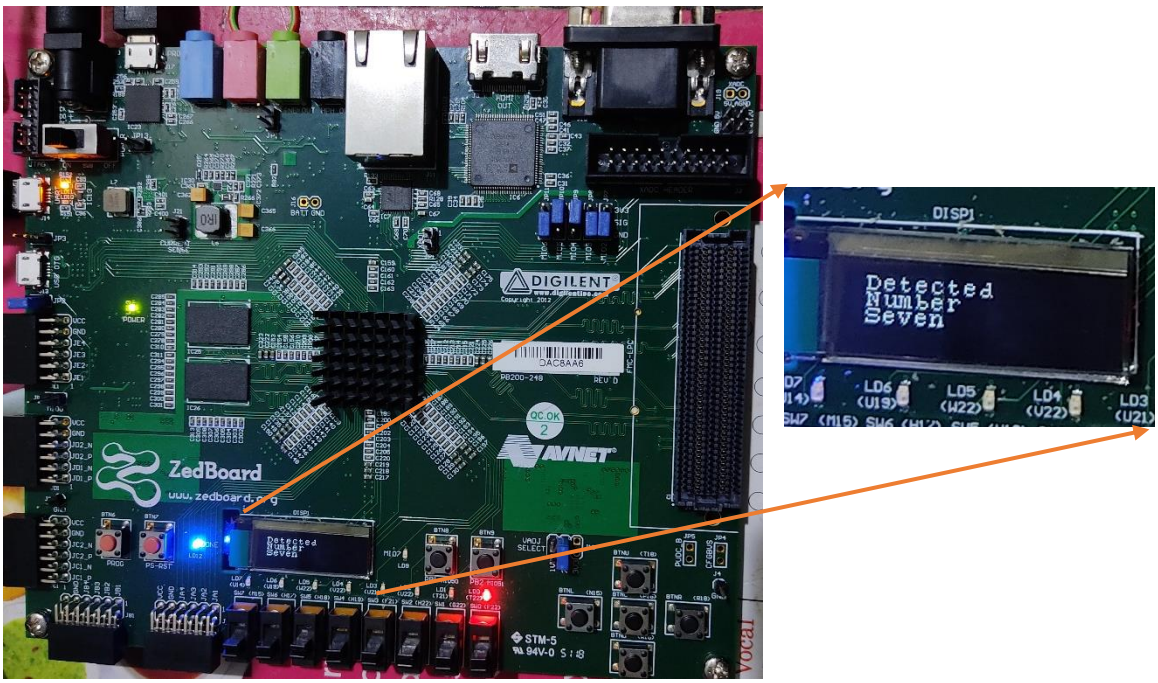


Figure 3.12: Output in FPGA Evaluation Kit

Chapter

4

Conclusions and Future Scope

4.1 Conclusions

This project on Multilayer Perceptrons (MLP) has been a journey of exploration, experimentation, and innovation in the realm of machine learning and FPGA-based acceleration. Through meticulous design, rigorous testing, and thoughtful analysis, endeavored to develop a robust and efficient MLP implementation tailored for deployment on FPGA platforms. Throughout this project, the effectiveness of MLPs in tackling diverse machine learning tasks, from classification to regression, leveraging their inherent flexibility and scalability. By harnessing the power of parallel processing and hardware acceleration offered by FPGAs, significant performance enhancements, accelerating both training and inference phases have been achieved maintaining high accuracy levels. This methodology, which combines Python programming for data generation, Verilog for hardware description, and C programming for system-level integration, exemplifies a holistic approach to FPGA-based ML development. By seamlessly integrating software and hardware components, the development process streamlined well-enough, enabling rapid prototyping and efficient iteration cycles. Furthermore, the comparative analysis against existing works underscores the competitiveness and efficiency of the proposed method. With superior accuracy, reduced resource utilization, and lower power consumption, the proposed MLP implementation stands out as a compelling solution for real-world deployment, particularly in resource-constrained environments. This proposed MLP implementation technique represents a significant milestone in the convergence of machine learning and hardware acceleration, offering promising avenues for innovation and impact in the fields of artificial intelligence and embedded systems.

4.2 Future Scope

While Multilayer Perceptrons (MLP) demonstrate remarkable potential in harnessing hardware resources efficiently, it's essential to acknowledge the continued dominance of Convolutional Neural Networks (CNN) in various applications. CNNs offer distinct advantages, particularly in tasks involving image processing and computer vision, where their hierarchical feature extraction capabilities shine. However, it's worth noting that the adopted framework in this work, ZyNet, is tailored specifically for MLP implementation and lacks the capability to efficiently accommodate CNN architectures.

Looking ahead, the future scope of this work lies in the development of a comprehensive framework capable of converting diverse neural network architectures, including CNNs, Recurrent Neural Networks (RNNs), and beyond, into Hardware Description Language (HDL) representations. This envisioned framework aims to inherit the versatility, efficiency, and user-friendly utilities embodied in ZyNet while extending its functionality to support a broader range of neural network paradigms.

By enabling the seamless conversion of various neural network models into hardware-accelerated implementations, this future framework would empower developers and researchers to utilize FPGA-based acceleration across a wider spectrum of machine learning tasks and applications. From real-time image recognition to natural language processing and beyond, the ability to deploy diverse neural network architectures efficiently on FPGA platforms promises to unlock new opportunities and drive innovation in embedded systems, edge computing, and beyond.

In essence, the future endeavors aspire to bridge the gap between neural network research and FPGA-based hardware acceleration, fostering a symbiotic relationship that makes use of the strengths of both domains to push the boundaries of performance, efficiency, and versatility in machine learning deployment.

References

[1] Y. Geng, X. Wang, and P. Jiang, “Prediction of the Cement Grate Cooler Pressure in the Cooling Process Based on a Multi-Model Fusion Neural Network,” *IEEE Access*, vol. 8, pp. 115028–115040, 2020, doi: 10.1109/ACCESS.2020.3002768.

[2] N. Jalodia, M. Taneja, and A. Davy, “A Deep Neural Network-Based Multi-Label Classifier for SLA Violation Prediction in a Latency Sensitive NFV Application,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 2469–2493, 2021, doi: 10.1109/OJCOMS.2021.3122844.

[3] F. Lin, “Supervised Learning in Neural Networks: Feedback-Network-Free Implementation and Biological Plausibility,” *IEEE Trans Neural Netw Learn Syst*, vol. 33, no. 12, pp. 7888–7898, 2022, doi: 10.1109/TNNLS.2021.3089134.

[4] T.-L. Huoh, Y. Luo, P. Li, and T. Zhang, “Flow-Based Encrypted Network Traffic Classification With Graph Neural Networks,” *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1224–1237, 2023, doi: 10.1109/TNSM.2022.3227500.

[5] Z. Huang *et al.*, “Convolutional Neural Network Based on Complex Networks for Brain Tumor Image Classification With a Modified Activation Function,” *IEEE Access*, vol. 8, pp. 89281–89290, 2020, doi: 10.1109/ACCESS.2020.2993618.

[6] Q. Xin, S. Hu, S. Liu, L. Zhao, and Y.-D. Zhang, “An Attention-Based Wavelet Convolution Neural Network for Epilepsy EEG Classification,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, pp. 957–966, 2022, doi: 10.1109/TNSRE.2022.3166181.

[7] X. Li and S. Wang, “Object Detection Using Convolutional Neural Networks in a Coarse-to-Fine Manner,” *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 11, pp. 2037–2041, 2017, doi: 10.1109/LGRS.2017.2749478.

[8] H. Yang, “Transmission Line Fault Detection Based on Multi-layer Perceptron,” in *2022 International Conference on Big Data, Information and Computer Network (BDICN)*, 2022, pp. 778–781. doi: 10.1109/BDICN55575.2022.00151.

[9] Y. Jusman, I. M. Firdiantika, D. A. Dharmawan, and K. Purwanto, “Performance of Multi Layer Perceptron and Deep Neural Networks in Skin Cancer Classification,” in *2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech)*, 2021, pp. 534–538. doi: 10.1109/LifeTech52111.2021.9391876.

[10] A. F. Kurniawan, Z. Luhur Pakerti, and G. F. Shidik, “Tune Up Multi Layer Perceptron From Sentiment Analysis with Natural Language Processing,” in *2023 International Seminar on Application for Technology of Information and Communication (iSemantic)*, 2023, pp. 112–116. doi: 10.1109/iSemantic59612.2023.10295311.

[11] G. S. V. S. Sivaram and H. Hermansky, “Sparse Multilayer Perceptron for Phoneme Recognition,” *IEEE Trans Audio Speech Lang Process*, vol. 20, no. 1, pp. 23–29, 2012, doi: 10.1109/TASL.2011.2129510.

[12] Y. Cheng, Y.-J. Deng, W.-Y. Wang, C.-F. Long, and X.-H. Zhu, “Locality Preserved MLP-Mixer for Hyperspectral Image Classification,” in *2023 13th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2023, pp. 1–5. doi: 10.1109/WHISPERS61460.2023.10430722.

[13] M. Jarrar, A. Kerkeni, A. Ben Abdallah, and M. H. Bedoui, “MLP Neural Network Classifier for Medical Image Segmentation,” in *2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV)*, 2016, pp. 88–93. doi: 10.1109/CGiV.2016.26.

[14] K. Vipin, “ZyNet: Automating Deep Neural Network Implementation on Low-Cost Reconfigurable Edge Computing Platforms,” in *Proceedings - 2019 International Conference on Field-Programmable Technology, ICFPT 2019*, Institute of Electrical and Electronics Engineers Inc., Dec. 2019, pp. 323–326. doi: 10.1109/ICFPT47387.2019.00058.

[15] M. Cho and Y. Kim, “Implementation of Data-optimized FPGA-based Accelerator for Convolutional Neural Network,” in *2020 International Conference on Electronics, Information, and Communication (ICEIC)*, 2020, pp. 1–2. doi: 10.1109/ICEIC49074.2020.9050993.

[16] H. Madadum and Y. Becerikli, “FPGA-Based Optimized Convolutional Neural Network Framework for Handwritten Digit Recognition,” in *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, 2019, pp. 1–6. doi: 10.1109/UBMYK48245.2019.8965628.

[17] T.-H. Tsai, Y.-C. Ho, and M.-H. Sheu, “Implementation of FPGA-based Accelerator for Deep Neural Networks,” in *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2019, pp. 1–4. doi: 10.1109/DDECS.2019.8724665.

[18] R. Xiao, J. Shi, and C. Zhang, “FPGA Implementation of CNN for Handwritten Digit Recognition,” in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2020, pp. 1128–1133. doi: 10.1109/ITNEC48623.2020.9085002.

[19] J. Si, E. Yfantis, and S. L. Harris, “A SS-CNN on an FPGA for Handwritten Digit Recognition,” in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2019, pp. 88–93. doi: 10.1109/UEMCON47517.2019.8992928.

[20] I. Westby, X. Yang, T. Liu, and H. Xu, “FPGA acceleration on a multi-layer perceptron neural network for digit recognition,” *Journal of Supercomputing*, vol. 77, no. 12, pp. 14356–14373, Dec. 2021, doi: 10.1007/s11227-021-03849-7.

[21] L. Deng, “The MNIST database of handwritten digit images for machine learning research,” *IEEE Signal Process Mag*, vol. 29, no. 6, pp. 141–142, 2012, doi: 10.1109/MSP.2012.2211477.