SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V11I2P111

Volume 11 Issue 2, 98-106, February 2024
© 2024 Seventh Sense Research Group®

Original Article

Designing A Low Power LeNet Convolutional Neural Network Accelerator for FPGA in IoT Edge Computing

Alshima Alwali[1], Peter K. Kihato[1,2]

[1]*Department of Electrical Engineering (Computer Option),*
Pan African University, Institute for Basic Sciences, Technology and Innovation (PAUSTI), Nairobi, Kenya.
[2]*Department of Electrical and Electronic Engineering, Jomo Kenyatta University of Agriculture and Technology (JKUAT), Nairobi, Kenya.*

[1]*Corresponding Author : alwali.alshima@students.jkuat.ac.ke*

Received: 21 November 2023 Revised: 14 December 2023 Accepted: 21 January 2024 Published: 16 February 2024

Abstract *- With an emphasis on the Xilinx Artix XC7A200T FPGA, in this paper, a Convolutional Neural Network (CNN) tailored explicitly for FPGA deployment is designed and implemented. The method adapts the LeNet-1 model using a hardware description language; this choice is motivated by the model's minimal size, making it suitable for edge computing devices. With its parameterized module structure, the architecture, known as 'LeNet,' offers significant flexibility and adaptability. The design focuses on the modular architecture and diversity of Processing Elements (PEs), crucial for parallel processing in computationally demanding CNN tasks. Convolutional, pooling, and fully connected layers are customized to leverage the FPGA's capabilities. Multiple filter banks are utilized for effective input processing and feature extraction. The pooling layers are specifically designed to reduce feature dimensionality complexity, thereby improving data fluctuation handling and reducing computational demands. The architecture stands out for its scalability and efficiency, utilizing five different processing units. The parameterization of modules and their successful application on the MNIST dataset, a standard benchmark in Machine Learning for handwritten digit recognition, further illustrate how the architecture may be adapted to different datasets and applications. The implementation of the Xilinx Artix XC7A200T FPGA achieved a power consumption of 1.775 W at 100 MHz, indicating that the design is energy-efficient and suitable for high-demand applications in resource-limited environments. This paper details the module design, parameterization, and integration methodologies employed in the design process of adapting the LeNet-1 model for FPGA.*

Keywords *- Convolutional Neural Networks, Edge computing, FPGA, LeNet-1, Performance analysis.*

1. Introduction

Many domains use Convolutional Neural Networks (CNNs), including semantic segmentation [1], object detection [2], and image classification [3]. Modern Convolutional Neural Nets (CNNs) include several layers and are more computationally complex, making them challenging to implement on embedded systems. In the domains of Automotive Driver Assistance (ADAS) and data center acceleration [4], using FPGA to speed CNN has garnered a lot of interest. FPGA's flexibility and ease of development allow it to accommodate the continually evolving CNN models.

In an attempt to attain high precision, CNNs have been trending toward layer additions, intricate architectures, and intricate operations [5-7]. The enormous volume of parameters and processes necessitates stringent memory management and computational power. First introduced in [9], a novel convolution design aimed to minimize the number of parameters and computing burden of standard convolutions. Xception [10], ShuffleNet [12], and MobileNetV2 [11] are examples of new CNNs that substitute depthwise separable convolution for traditional convolution. In order to retain a high degree of accuracy, this significantly reduces the number of processes and parameters required.

CNN is a computationally demanding task that uses a lot of processing power. The target platform is chosen to be the Graphic Processing Units (GPUs) because of their sufficient performance. However, GPUs have a significant issue due to power consumption. The highly parallel, scalable, and energy-efficient computing substrate of FPGAs is driving up their use for CNN acceleration [13].

There is still a significant difference between the CNN model and accelerator design, even with recent attempts to employ FPGAs to accelerate CNNs [14]. Large and ineffective standard CNN models like AlexNet [15], VGG16 [5], GoogLeNet [16], and ResNet [8] are still targeted at some

FPGA accelerators. More computational and storage resources are needed for those inefficient models than for the compact, accurate models that are efficient. Getting high performance out of an accelerator based on these inadequate approaches is challenging.

Modern CNN-based accelerators like MobileNets and ShuffleNets have only been able to reach respectable speeds on image recognition jobs. Depthwise separable convolution is usually used by these state-of-the-art CNNs in order to reduce parameters and operations.

There are numerous types of systems available in the field of AI/ML accelerators, ranging from modest designs to large-scale business solutions, all of which are customized to meet particular computing requirements.

In [17], Google's TPU project aimed to reduce profound neural network inference costs by 10x. It was developed as a CPU and GPU ASIC replacement between 2013 and 2015. This was done with systolic array-based hardware matrix multiplication units. The original TPU, known as TPU v1, used a 256 × 256 systolic array matrix multiplication unit for some workloads and provided 30-80× more performance per watt than current CPUs and GPUs.

Since then, TPU v2 and v3 have been launched. The TPU versions 2 and 3 are designed for inference and training and perform better than their predecessors. Bfloat16, a new number format introduced in TPU v2, has the same dynamic range as fp32 but poorer accuracy. Many Google products use TPUs, including Search, Android, YouTube, and others [17]. Microsoft Project Brainwave [18] is a machine-learning accelerator.

In contrast to Google TPU, Project Brainwave uses an FPGA-Project Brainwave benefits from being programmable and more agile than Google. In the study [22], Programmable Logic (PL) is employed for speed and power consumption, while a Processor System (PS) is used for process control. This method integrates streams of training and inference data by using High-Level Synthesis (HLS) tools to translate high-level language descriptions into hardware Register Transfer Level (RTL) formats.

Validation on the Xilinx HA device with the MNIST dataset showed remarkable processing speeds, low power consumption, and good accuracy, along with comparable convergence rates to GPU models in only 78.04% of the training period. For IoT edge computing, the B. Wang et al. "Shenjing" accelerator [19] is a low-power, compact device. By connecting a regular neural network to a spiking neural network, SNN energy efficiency is possible.

This approach uses 1.26 mW, and MNIST inference errors are 4%. "FANN-on-MCU" is another small neural network accelerator [20] developed by X. Wang et al. for edge computing in the Internet of Things. They used PULP, a parallel ultralow-power RISC-V platform, to code their approach, unlike previous research. Their architecture is 22x quicker and 69% less energy-hungry than Cortex-M4.

In [25], SparkNet, a lightweight neural network architecture, is created to decrease weight parameters and computing demands on MINIST, CIFAR-10, CIFAR-100, and SVHN datasets. Squeezenet's minimal parameters and deep separable convolution's reduced processing help SparkNet compress CNN 150x. SparkNOC, an FPGA-based accelerator architecture, maps each network layer to dedicated hardware units for pipelined operation.

SparkNOC, implemented on the Intel Arria 10 GX1150 FPGA platform, achieves high speed with convolutional layer parallelism, on-chip RAM for intermediate result access, and synchronized pipeline execution. In experiments, the accelerator earned 337.2 GOP/s with 44.48 GOP/s/W energy efficiency, beating GPU, CPU, and FPGA approaches.

Developing an efficient SparkNet architecture for edge deployment, a parallelism allocation technique for FPGA-based accelerators, and deploying SparkNet on FPGA with higher performance and power efficiency are the main contributions."

This study presents a new FPGA-based accelerator solution for the LeNet-1 Convolutional Neural Network, utilizing the Verilog hardware description language. This study aims to efficiently implement LeNet-1, a comparatively more straightforward CNN model crucial for edge computing applications.

In contrast to earlier efforts focusing on large-scale solutions or specific architectures like Google's TPU [17] and Microsoft Project Brainwave [18], this research introduces a parameterized LeNet module. This module features fully connected and convolutional layers that are meticulously instantiated, with adjustable filter sizes, channel counts, and pooling layer parameters.

The uniqueness of this approach lies in the combination of Processing Element (PE) and PE array modules optimized for parallel processing. This results in a significant increase in computational performance quantified in nanoseconds per processing step.

Suited for real-time image recognition in IoT edge computing environments, this approach not only addresses the challenges of implementing CNNs on FPGAs but also marks a notable improvement in executing computationally intensive tasks. By bridging the current gap in FPGA-based CNN accelerators, this work offers a scalable and energy-efficient solution applicable in various technological contexts.

2. Background

2.1. CNN Foundations

The 1980s and 1990s saw the start of Convolutional Neural Network research. The first convolutional neural networks were Lenet-5 and the time delay network. Convolutional Neural Networks were developed quickly around the turn of the 20th century, thanks to advancements in numerical computing hardware and deep learning theory. They are currently widely employed in several domains, including computer vision and natural language processing [15].

Bionic creatures' mechanisms for visual perception are built using Convolutional Neural Networks. The Convolutional Neural Network organizes the layer data with less processing, making feature extraction easier. The hidden layer's convolution kernel parameters and the absence of links between network layers make this possible. The feedforward portion of convolutional neural networks is frequently employed for picture recognition and classification, and the feedback section is used for network training when they are utilized as supervised learning techniques. The majority of users employ Convolutional Neural Networks and training weight data to accomplish real-time jobs; hence, feedforward calculation speed is more crucial. This article's primary focus is on the FPGA's feedforward section acceleration in convolutional neural networks.

The feature extractor and classifier are the two components that make up the conventional neural network architecture. The function of the feature extractor is to extract the features from the input image. Since the feature map was mapped to a different feature map, these images lack unique properties. This is primarily because of the convolution core sliding that contains most shapes.

The weight of the convolution kernel should be mapped one-to-one between the input and output layers. A classifier is what is created when the feature layer is transformed into an output structure. Feature extractors frequently include convolution and downsampling layers, like with Lenet-1, as seen in Figure 1. One fully connected layer, two subsampling layers, and two convolutional layers make up LeNet-1.

2.2. Acceleration of FPGA

FPGA clocks typically operate at a few hundred MHz, although the core frequency of a general-purpose CPU can reach several GHz. However, the general-purpose CPU typically performs better than the FPGA. For specific tasks, including signal or image processing, a general-purpose CPU may need a lot of clock cycles. However, through programming, an FPGA may immediately build a specific circuit. During this process, the logic blocks and connections on the FPGA are configured to maximize pipelining and parallel processes.

Furthermore, through memory use optimization, an FPGA speeds up reading and writing processes, resulting in a notable improvement in performance for particular tasks.

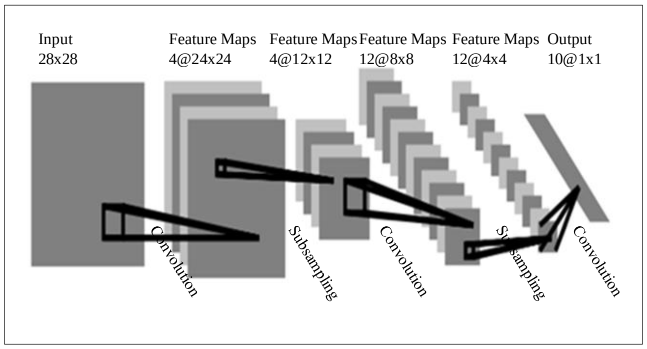


Fig. 1 Architecture of LeNet-1 [21]

Recent developments in FPGA optimization for faster processing rates have mainly concentrated on the creative creation and usage of practical algorithms and designs, as the LeNet model proposed in this work serves as an example. With skillful configuration, this Convolutional Neural Network takes advantage of FPGAs' parallel processing capabilities. A modular design, which divides the model into convolutional, pooling, and fully connected layers, is one of the main optimization techniques.

This allows for parallel processing while streamlining data flow and computation management. The model's pipelined architecture, which enables the simultaneous execution of numerous calculation steps, dramatically reduces latency and increases throughput. The FPGA's strategic memory management lets you get to weights and intermediate data quickly. It also cuts down on latency and speeds up convolutional processing by cutting down on data retrieval times.

The LeNet model is characterized by its parameterization, which lets you change things like the number and size of filters and the number of neurons in fully linked layers. This makes it scalable and adaptable to different FPGA sizes and capabilities. By using the same hardware resources for many layers at other times, sharing mechanisms allow for resource optimization through optimal resource use and reduced hardware requirements. To speed up processing, the model also uses fixed-point arithmetic instead of floating-point, which makes calculations easier and saves resources.

3. Accelerator Design

An extensive design overview of a hardware-accelerated Convolutional Neural Network (CNN) aimed at practical and high-performance computing applications is presented in this research. The design features a stable architecture, as shown in Figure 2, which integrates all the essential components of a CNN onto an FPGA platform. This integration resolves significant problems with real-time processing, power efficiency, and data management, making it practical for a variety of uses, including real-time image analysis and autonomous systems.

3.1. Architectural Design

The Processing Elements (PEs) are essential to the convolutional layer's effectiveness.

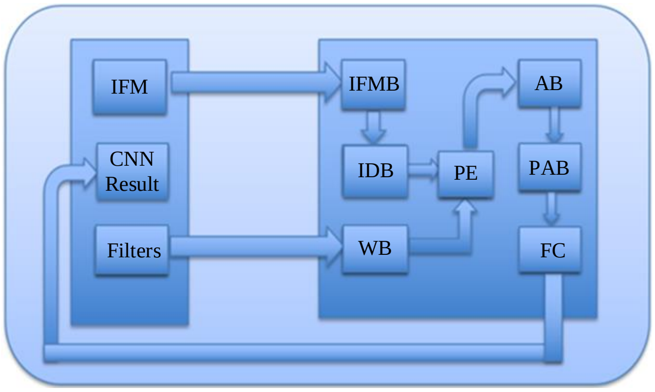


Fig. 2 Overview of accelerator design

These PEs are essential to the architecture's ability to process data in parallel. Fast matrix multiplication, a crucial process in convolutional computations, is a skill that each PE possesses. The solution allows for the simultaneous processing of several data streams by arranging a total of five PEs in an array. This arrangement offers a scalable and flexible design that supports many CNN models while also speeding up convolutional processes. For quick feature extraction in real-time applications, the PEs are tuned for low-latency operations. Their modular architecture improves the reconfigurability and adaptability of the system to meet a range of computational needs. Significant gains in computational performance are achieved, along with the preservation of power economy, by integrating these Processing Elements (PEs)-a crucial feature for FPGA-based devices.

By utilizing these PEs in a Processing Element Array, the convolutional layers improve matrix multiplication performance and data flow. After convolution, the Accumulation Block (AB), Figure 3, consists of an adder to add the incoming partial sum to its corresponding counterpart and a FIFO to store partial sums previously kept in that block. The time-synchronized addition process keeps on until every block contains the entire convolved result. When no pertinent data arrives from the systolic array, the contents of each block are frozen. The Pooling and Activation Block (PAB), Figure 3, is made up of an activation unit, a memory element (FIFO), and a comparator block, then processes the data. The following inputs from the accumulation block are compared to the matching item already kept in this block's residual FIFO, and the most significant value in the comparator block's output is one of these values. PAB employs an enhanced Softmax Activation function alongside a specially designed pooling method tailored to meet the requirements of the application. This approach effectively reduces spatial dimensions and introduces non-linearity. This strengthens the network's ability to identify complex patterns.

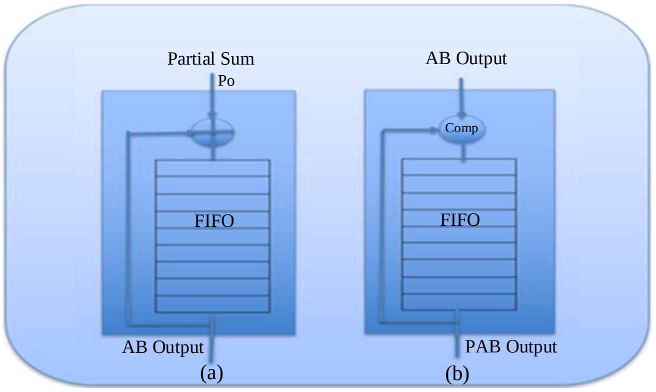


Fig. 3 (a) AB module, and (b) PAB module.

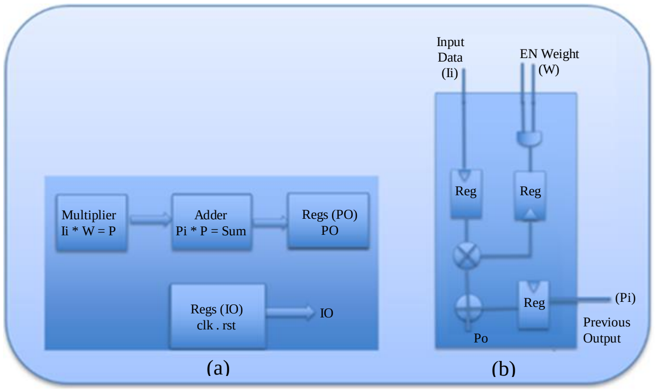


Fig. 4 (a) PE dataflow, and (b) PE architecture.

The Fully Connected (FC) module in Figure 2 is essential for combining information and enabling classification. The FC serves as a point of convergence for the features retrieved by the convolutional and pooling layers that came before it, synthesizing the data into a format that makes sense for the decision-making process. Its primary function is to classify the input image by allocating it to one of the predefined classes (0-9), especially in digit recognition tasks. Multiply-accumulate activities, in which each neuron evaluates its inputs against corresponding weights to produce a collection of outputs, are the fundamental operations of the FC module. A softmax function, which is common in classification problems, is then used to convert these outputs into probabilities, yielding an understandable and straightforward conclusion. To further enhance the layer's effective and well-organized operation, the code incorporates well-considered control logic that manages reset circumstances, regulates the sequence in which operations are carried out, and indicates when processing is complete.

3.2. Processing Element

The critical element of the proposed FPGA-based Convolutional Neural Network (CNN) architecture is the Processing Element (PE) module, which is seen in Figure 4. The PE Array module (Figure 5) successfully complements the PE module in this design. The multiplier and adder blocks, which carry out the primary computational operations, are what define the PE module. Weights are applied to the input data (Ii) in the multiplier block in order to get the intermediate product (P). In the Adder Block, this product is joined with Pi, the previous output, to create a total, which effectively integrates the outcomes of the last and current processes.

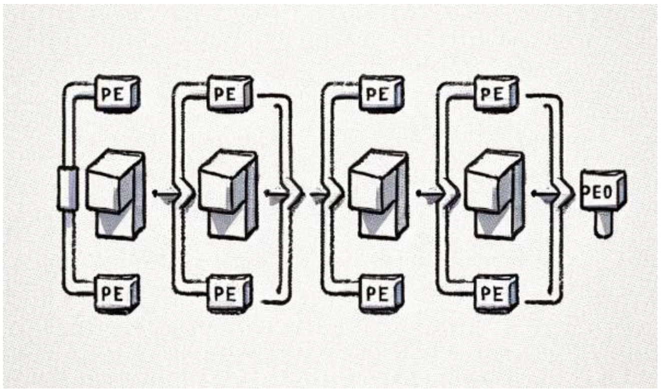


Fig. 5 PE array

Two sets of registers are used in the PE module to control data flow and synchronization. The clk (clock) signal ensures timely processing, while the reset (rst) signal provides initialization. The first set, Registers (Po), records the output from the Adder Block. Po, the output from these registers, represents one significant computational outcome. Ii directly feeds a second set, Registers (Io), which is governed by clk and rst. The synchronized version of the input data that these registers output is called Io, and it is essential for further system activities as well as for maintaining data integrity and coherence with the module's processing cycle.

An essential part of the architecture is the "PE Array" module, which is a skillfully constructed array of Processing Elements (PEs) that are necessary for convolutional operations in neural networks. Each PE acts on a different set of input data and weights. A very straightforward diagram of this module (Figure 5) shows five connected rectangles with structured labels, PE0 through PE4. Every rectangle, which stands for a PE, is connected to provide smooth data flow, which is essential for activities requiring sequential processing. According to this architecture, each PE in the array processes the inputs in a certain way before passing the processed data to the next PE in line. This design, which is visible in the schematic as well as the practical implementation, illustrates a sequential data processing system. A series of processing steps is formed when the output of one PE is used as the input for the following.

Moreover, the PE Array module can be customized to fulfill unique application requirements and accommodate different bit widths. The PEs' interconnectedness, which results in one PE's output becoming another's input, creates a seamless data flow that is necessary for sequential processing. Intermediary wires help this flow, and the array's general configuration guarantees effective data transfer between the PEs. The array's embedded clock and reset signals allow for operation resetting and synchronization, which improves functionality overall.

This ingenious approach, which is especially well-suited for the convolutional stages of neural networks, finds a balance between the parallel processing powers of FPGA structures and sequential data handling. The combination of the PE and PE Array modules in the architecture provides a high-performance computation technique for neural network applications by fully utilizing FPGA technology. Essentially, the PE Array module effectively communicates the concept of complex computing processes common to sophisticated digital systems, as well as serving as an example of a basic idea in the domains of digital signal processing and neural networks. Though it seems straightforward, this method captures the core of practical, step-by-step data processing, where each component builds on the work of the one before it.

3.3. Memory Management

As Figure 2 illustrates, on-chip memory is essential to improving system performance in the CNN accelerator. The WB is used to store the filter weights required for convolution operations. It is made to efficiently keep weights in a way that facilitates high-throughput access, which is crucial for quick convolution computations. The input feature maps, on the other hand, are either the original input data or the outputs of earlier layers and are stored in the IFMB. Taking into consideration that feature maps alter as processing moves forward, it is tuned for dynamic read/write operations. These two elements are essential for decreasing off-chip memory transfer delay and increasing data access speed. This is crucial for FPGA systems in particular because memory bandwidth is often a limiting factor. These buffers significantly improve system performance by localizing important data inside the FPGA, facilitating quick and effective computation that is essential for real-time processing applications.

The Intermediate Data Buffer (IDB) is also integrated into the design, which improves the data processing flow even further. The IDB sequences data from the IFMB before sending it to the Processing Elements (PEs). To provide consistent data transport, each row of the IDB in this configuration corresponds to a row of the processing elements. The data traversal order, which essentially determines how the data is fed into the PEs, is carefully followed when organizing the data from the IFMB in the IDB.

This configuration improves the use of the processing elements and simplifies the data flow from IFMB to PEs, improving the effectiveness and performance of the CNN accelerator. The output Feature Map, or FM, is moved to off-chip DRAM after first being kept in on-chip buffers. When using deep learning models, this approach is crucial because the network produces large volumes of data in these models. For processing or storing, effective data management is essential to avoiding overstuffing the device's memory and bandwidth.

Dynamic weight loading, efficient data compression, and selective input data tiling are incorporated to enhance the effectiveness of the concept further. These techniques are crucial for getting over the built-in restrictions on bandwidth and on-chip memory, guaranteeing that the architecture can balance throughput, latency, and power efficiency while meeting the demanding computational requirements of contemporary Deep Learning applications. Maintaining this equilibrium is essential for the accelerator's incorporation into more extensive systems, as every one of these elements is critical to overall functionality.

4. Results and Discussion

This research utilized the Vivado software suite for the design and synthesis of an FPGA-based CNN accelerator. Key to the testing was the MNIST dataset, renowned in Machine Learning for handwritten digit recognition. This choice ensured the architecture's applicability in real-world image processing and neural network applications. The design was rigorously tested using the high-performance Xilinx Artix XC7A200T FPGA, known for its adaptability and cost-efficiency. This platform provided the necessary environment for reliable testing of functionality and performance.

Figure 6 illustrates the implemented design within Vivado, highlighting the layout and configuration of various components. The design's resource usage is broken down in Table 1. Notably, the total power consumption was measured at 1.775W at 100MHz, as shown in Figure 7. This consumption, comprising 1.638 W of dynamic power and 137 mW of static energy, was estimated using the Xilinx Power Estimator (XPE). Remarkably, this represents a 55% reduction in power compared to previous LeNet CNN designs, which typically consumed around 3.22 W.

The FPGA CNN accelerator's output underwent rigorous verification against Python-based implementations, showcasing minor differences attributable to distinct precision handling and computing methods between the platforms, as illustrated in Figure 8. Despite these variances, the design demonstrated remarkable flexibility and resilience across a variety of processing scenarios. The comparative analysis evaluates the FPGA-based CNN accelerator against previous models [23, 24], which also focus on LeNet CNN architectures but employ different design strategies. A fixed-point data format and numerous approximate accumulation units were used in the work [23] to propose an FPGA-based CNN accelerator.

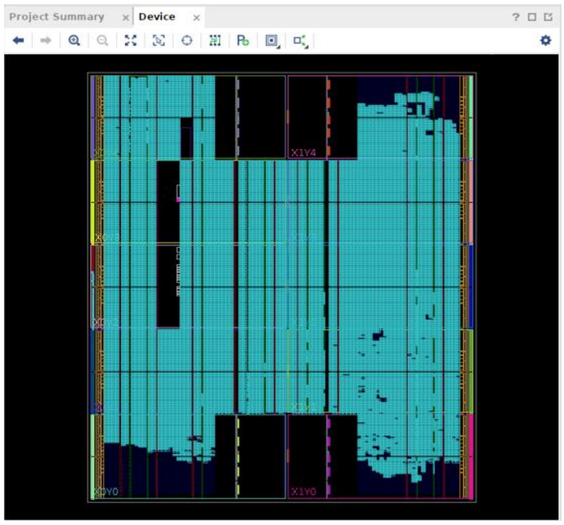
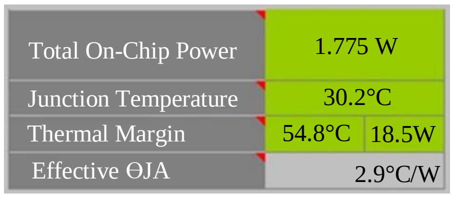


Fig. 6 Hardware implementation in Vivado



Total On-Chip Power	1.775 W	
Junction Temperature	30.2°C	
Thermal Margin	54.8°C	18.5W
Effective ΘJA	2.9°C/W	

Fig. 7 Power summary

This design, developed using high-level synthesis tools on a Xilinx FPGA, optimized memory usage and network latency by 66% and 50%, respectively, compared to floating-point designs. It optimized data types and loop parallelization and used FPGA logic resources for approximation operations to achieve this efficiency. Their approach included an approximate MAC operator and data size optimization by trimming unused bits post-activation, with performance tested across various bit widths.

On the other hand, [24] developed a CNN accelerator for FPGA with the goal of identifying handwritten numbers in MNIST. Their system utilized deep pipeline processing to optimize parallelism at both coarse and fine granularity levels. The design featured a structured circuit approach for easy expansion of layers and neurons and improved classification throughput by efficient internal memory organization in the FPGA. This approach resulted in three times the acceleration at 50MHz compared to traditional CPUs, with power consumption taking up only 2% of CPU usage. They emphasized a flexible memory management system, standard interfaces for convolution and pooling layers, and a structured design enabling easy CNN reconstruction and scalability.

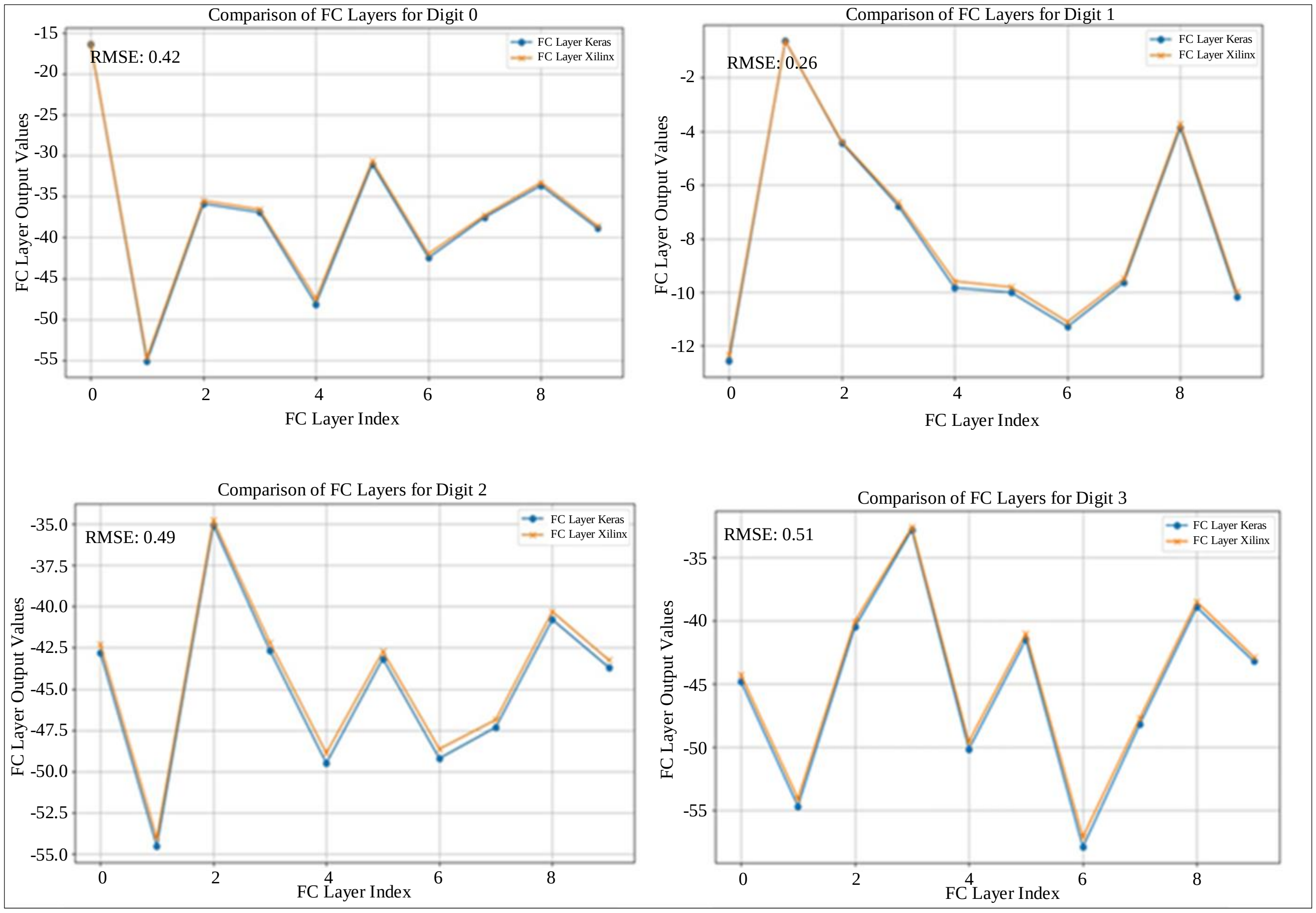


Fig. 8 Samples of contrasting XILINX and KERAS outputs

Table 1. Comparison of the use of FPGA resources

Model	**This Work**	**[23]**	**[24]**
FPGA	Artix-7XC7A200T	ZynqXCZU9EG	Artix-7XC7A20
Clock (MHz)	100	100	50
LUT	72,886	61,713	88,756
FF	36,252	27,863	42,038
DSP	141	123	571
IO	34	-	-
Power (W)	1.775	1.673	14.13

The design, while not employing advanced techniques like deep pipeline processing or structured circuit design, still demonstrated superior performance compared to [24] and was nearly equivalent to [23]. As detailed in Table 1, the approach focused on effective resource utilization without the use of high-precision DSP blocks. It achieved notable improvements in memory usage, network latency, and power efficiency. This comparison highlights the design's strength in delivering high performance and energy efficiency, validating it as a competitive solution in the FPGA-based CNN accelerator landscape.These comparisons and test results underscore the efficiency of the FPGA-based CNN accelerator design, particularly in terms of power consumption and performance. The approach effectively combines optimized data formats and parallelization strategies, presenting a significant advancement in the field of FPGA-based neural network implementation, especially suited for applications where power efficiency and processing speed are paramount.

5. Conclusion

An FPGA-based CNN architecture accelerator designed especially for the LeNet-1 architecture is implemented in this work. The design is notable for using Vitis HLS in Vivado to construct the layers and overall structure of the CNN accelerator, achieving an accuracy of over 96%. Based on the FPGA Xilinx Artix-7 XC7A200T, the design has been tested. Amazingly, the accelerator achieves the needed throughput at 100 MHz with just 1.775 W of power consumption. Accordingly, in terms of performance per watt, the suggested approach performs better than current LeNet FPGA implementations. Aside from these successes, the architecture has also been effectively used with the MNIST dataset, a standard benchmark for assessing Machine Learning models when it comes to handwritten digit recognition. This modification highlights the approach's adaptability and potency even further. Future improvements to the performance assessment methodology will encompass the incorporation of parallel data transfer for both the Input Feature Map (IFM) and weights, permitting data accessibility within a single clock cycle, and providing internal storage to concurrently store the weights and biases. This will reduce memory bottlenecks and improve the functionality of the solution in CNN-embedded real-time applications.

Funding Statement

This study was supported by the African Union (AU).

References

[1] Kaiming He et al., "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2961-2969, 2017. [Google Scholar] [Publisher Link]

[2] Shaoqing Ren et al., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Advances in Neural Information Processing Systems*, vol. 28, pp. 1-9, 2015. [Google Scholar] [Publisher Link]

[3] Olga Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211-252, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[4] Andrew Putnam et al., "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13-24, 2014. [Google Scholar] [Publisher Link]

[5] Karen Simonyan, and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *ArXiv*, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[6] Christian Szegedy et al., "Rethinking the Inception Architecture for Computer Vision," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818-2826, 2016. [Google Scholar] [Publisher Link]

[7] Christian Szegedy et al., "Inception-v4, Inception-Resnet and the Impact of Residual Connections on Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, pp. 4278-4284, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[8] Kaiming He et al., "Deep Residual Learning for Image Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, 2016. [Google Scholar] [Publisher Link]

[9] Laurent Sifre, and Stephane Mallat, "Rigid-Motion Scattering for Texture Classification," *ArXiv*, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[10] Francois Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1251-1258, 2017. [Google Scholar] [Publisher Link]

[11] Mark Sandler et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510-4520, 2018. [Google Scholar] [Publisher Link]

[12] Xiangyu Zhang et al., "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6848-6856, 2018. [Google Scholar] [Publisher Link]

[13] Yongming Shen, Michael Ferdman, and Peter Milder, "Maximizing CNN Accelerator Efficiency through Resource Partitioning," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 535-547, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[14] Kiseok Kwon et al., "Co-Design of Deep Neural Nets and Neural Net Accelerators for Embedded Vision Applications," *DAC '18: Proceedings of the 55th Annual Design Automation Conference*, pp. 1-6, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, pp. 1-9, 2012. [Google Scholar] [Publisher Link]

[16] Christian Szegedy et al., "Going Deeper with Convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-9, 2015. [Google Scholar] [Publisher Link]

[17] Norman P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," *Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1-12, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[18] Jeremy Fowers et al., "A Configurable Cloud-Scale DNN Processor for Real-Time AI," *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Los Angeles, USA, pp. 1-14, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[19] Xiaying Wang et al., "FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403-4417, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[20] Chen Zhang et al., "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks," *FPGA '15: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161-170, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[21] Yann LeCun et al., "Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition," *Neural Networks: The Statistical Mechanics Perspective*, vol. 2, pp. 261-276, 1995. [Google Scholar]

[22] Tianling Li, Bin He, and Yangyang Zheng, "Research and Implementation of High Computational Power for Training and Inference of Convolutional Neural Networks," *Applied Sciences*, vol. 13, no. 2, pp. 1-20, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[23] Mannhee Cho, and Youngmin Kim, "FPGA-Based Convolutional Neural Network Accelerator with Resource-Optimized Approximate Multiply-Accumulate Unit," *Electronics*, vol. 10, no. 22, pp. 1-16, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[24] Dan Shan, Guotao Cong, and Wei Lu, "A CNN Accelerator on FPGA with A Flexible Structure," *2020 5th International Conference on Computational Intelligence and Applications (ICCIA)*, Beijing, China, pp. 211-216, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[25] Ming Xia et al., "SparkNoC: An Energy-Efficiency FPGA-Based Accelerator Using Optimized Lightweight CNN for Edge Computing," *Journal of Systems Architecture*, vol. 115, 2021. [CrossRef] [Google Scholar] [Publisher Link]