International Journal for Modern Trends in Science and Technology
Volume 12, Issue 07, pages 281-288.
ISSN: 2455-3778 online
Available online at: http://www.ijmtst.com/vol12issue06.html
DOI: https://doi.org/10.5281/zenodo.21706560

Design and Implementation of a Systolic Array-Based CNN Accelerator Using Verilog HDL for Handwritten Digit Recognition

Achintya Surya Veedhinav Magapu | Y V Bhaskar Reddy | S Arunasri

Department of Electronics and Communication Engineering, QIS College of Engineering and Technology, Vengamukkapalem, Andhra Pradesh, India.

To Cite this Article
Achintya Surya Veedhinav Magapu, Y V Bhaskar Reddy & S Arunasri (2026). Design and Implementation of a Systolic Array-Based CNN Accelerator Using Verilog HDL for Handwritten Digit Recognition. International Journal for Modern Trends in Science and Technology, 12(07), 281-288. https://doi.org/10.5281/zenodo.21706560

Article Info
Received: 07 July 2026; Revised: 23 July 2026; Accepted: 26 July 2026.

KEYWORDS

Convolutional Neural Network (CNN), LeNet-1, FPGA, Systolic Array, Verilog HDL, Hardware Accelerator, Edge Computing, Parallel Processing, Image Classification, Vivado, MNIST.

ABSTRACT

Convolutional Neural Networks (CNNs) have become one of the most effective techniques for image classification and pattern recognition. However, implementing CNN inference on general-purpose processors often results in increased computational complexity, memory bandwidth requirements, and higher power consumption, making them less suitable for real-time embedded applications. Field-Programmable Gate Arrays (FPGAs) provide an attractive alternative by offering parallel processing capabilities, low latency, and energy-efficient hardware acceleration. This work presents the design and implementation of a systolic array-based LeNet-1 Convolutional Neural Network accelerator for handwritten digit recognition on an FPGA platform. The proposed framework combines software-based model training using the MNIST dataset with a hardware implementation developed in Verilog HDL. The trained network parameters are extracted and converted into memory initialization files, enabling direct deployment on the FPGA. The hardware architecture consists of convolution, pooling, fully connected, and softmax modules integrated through a systolic array of processing elements to maximize data reuse and computational parallelism. Functional verification is performed using Vivado simulation, and the hardware outputs are compared with the software-generated reference outputs to validate the correctness of the accelerator. The proposed architecture demonstrates an efficient hardware–software co-design approach for CNN inference while maintaining low resource utilization, reduced execution latency, and suitability for resource-constrained edge computing applications.

I. INTRODUCTION

The rapid advancement of Artificial Intelligence (AI) has significantly transformed modern computing systems, particularly in computer vision, pattern recognition, autonomous systems, robotics, and Internet of Things (IoT) applications. Among various AI models, Convolutional Neural Networks (CNNs) have emerged as one of the most successful deep learning techniques due to their exceptional performance in image classification and feature extraction tasks. Since the pioneering work of LeCun *et al.* on handwritten digit recognition, CNNs have demonstrated remarkable accuracy while eliminating the need for handcrafted feature extraction methods [1], [2]. More recently, deeper CNN architectures have further improved classification accuracy across large-scale datasets, establishing CNNs as the foundation of many state-of-the-art vision systems [3].

Despite their superior performance, CNN inference requires a large number of multiply-accumulate (MAC) operations, extensive memory accesses, and high computational throughput. These requirements make software-based implementations on conventional processors inefficient for real-time embedded applications due to increased execution latency and power consumption [4], [8]. As AI applications continue to migrate toward edge computing devices, there is an increasing demand for hardware platforms capable of executing CNN models efficiently while maintaining low power consumption and minimal hardware resources [6].

Field-Programmable Gate Arrays (FPGAs) have emerged as a promising platform for accelerating deep learning applications because of their inherent parallelism, reconfigurable architecture, and energy-efficient operation. Unlike Graphics Processing Units (GPUs), FPGAs allow designers to develop customized hardware architectures that maximize parallel execution while reducing memory bottlenecks and latency [4], [5]. Their flexibility makes them particularly suitable for implementing CNN inference engines in resource-constrained embedded and edge computing environments.

Among various hardware acceleration techniques, systolic array architectures have gained significant attention due to their ability to efficiently perform repetitive matrix multiplication and convolution operations. A systolic array consists of multiple Processing Elements (PEs) that communicate locally with neighboring elements, enabling continuous data flow, high computational throughput, and effective data reuse. These characteristics significantly reduce external memory accesses and improve hardware utilization during CNN inference [5], [6]. Consequently, systolic array-based architectures have become one of the preferred solutions for FPGA-based CNN accelerators.

The LeNet-1 convolutional neural network provides an excellent benchmark for studying FPGA-based CNN acceleration because of its relatively simple architecture while preserving the essential computational characteristics of modern convolutional networks. It consists of convolution, pooling, fully connected, and output layers that collectively perform handwritten digit classification on the MNIST dataset [2]. Owing to its modest computational complexity, LeNet-1 enables efficient hardware realization while serving as an ideal platform for validating FPGA accelerator architectures.

In this work, a **systolic array-based LeNet-1 CNN accelerator** is designed and implemented using Verilog HDL for FPGA deployment. The CNN model is first trained using the MNIST handwritten digit dataset in Python, after which the trained parameters are extracted and converted into memory initialization files suitable for FPGA implementation. The proposed hardware architecture comprises dedicated convolution, pooling, fully connected, and softmax modules interconnected through a systolic array of processing elements to exploit parallel computation and data reuse. Functional verification is performed using Vivado simulation by comparing the FPGA-generated outputs with the corresponding software-generated reference results. The complete framework demonstrates an efficient hardware–software co-design methodology for CNN inference that is suitable for low-cost FPGA platforms and resource-constrained edge computing applications. The implementation closely follows the open-source LeNet FPGA accelerator architecture while providing a systematic workflow from software model training to hardware realization and verification.

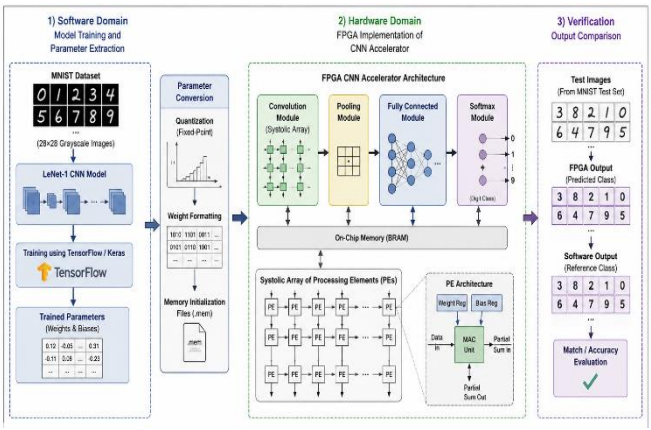

Fig. 1 Overall framework of the proposed CNN accelerator on FPGA

Fig. 1 illustrates the overall workflow of the proposed LeNet-1 CNN accelerator, integrating software-based model training, parameter extraction, FPGA hardware implementation, and output verification. The proposed framework employs a systolic array-based architecture to accelerate convolution operations, enabling efficient CNN inference through parallel processing and optimized data reuse on FPGA platforms.

The major contributions of this work are summarized as follows:

- Design and training of a LeNet-1 CNN model for handwritten digit recognition using the MNIST dataset.
- Extraction and conversion of trained network parameters into FPGA-compatible memory initialization files.
- Development of a systolic array-based CNN accelerator using Verilog HDL for efficient convolution processing.
- Integration of convolution, pooling, fully connected, and softmax modules into a complete FPGA inference pipeline.
- Functional verification of the hardware accelerator through Vivado simulation and comparison with software-generated outputs.
- Demonstration of an efficient hardware–software co-design framework for FPGA-based CNN acceleration suitable for embedded and edge AI applications.

II. RELEATED WORK

The increasing computational complexity of Convolutional Neural Networks (CNNs) has motivated extensive research on hardware acceleration techniques capable of delivering high throughput while maintaining low power consumption. Among various hardware platforms, Field-Programmable Gate Arrays (FPGAs) have become a preferred choice because of their inherent parallelism, reconfigurability, and ability to implement application-specific architectures for deep learning inference [4], [5].

The development of CNNs began with the pioneering work of LeCun et al., who introduced the LeNet architecture for handwritten digit recognition using backpropagation learning. Their work demonstrated that convolutional neural networks could automatically learn hierarchical image features, eliminating the need for manually engineered feature extraction methods [1]. Later, LeCun *et al.* further improved this architecture by proposing LeNet-5, which became one of the earliest practical CNN models for document recognition and established the foundation of modern deep learning-based image classification [2].

The remarkable success of deep CNNs was further demonstrated by Krizhevsky et al. through AlexNet, which significantly outperformed conventional image classification techniques on the ImageNet dataset and inspired widespread adoption of deep learning for computer vision applications [3]. Although deeper CNN architectures provide higher classification accuracy, they also introduce substantial computational complexity and increased memory requirements, making efficient hardware acceleration an essential research topic.

Several researchers have investigated FPGA-based CNN accelerators to improve inference performance while reducing energy consumption. Farabet et al. proposed one of the earliest FPGA processors dedicated to convolutional neural networks, demonstrating that specialized hardware architectures can significantly accelerate convolution operations compared with software implementations [10]. Similarly, Cadambi et al. introduced a programmable parallel accelerator capable of supporting both CNN training and inference by exploiting data-level parallelism within convolution operations [9].

Since memory access often dominates CNN execution time, Peemen et al. proposed a memory-centric accelerator architecture that maximizes data reuse by optimizing the movement of feature maps and filter weights between on-chip and off-chip memories [8]. Their work highlighted that efficient memory

management is equally important as computational optimization when designing FPGA-based CNN accelerators.

To further improve computational throughput, Zhang et al. introduced an optimized FPGA accelerator design that applies loop tiling and parallel processing techniques for deep convolutional neural networks [4]. Their architecture demonstrated considerable improvements in throughput by carefully balancing computation and memory bandwidth. Building upon this idea, Wei et al. proposed an automated synthesis methodology for systolic array architectures that generates high-performance FPGA implementations by exploring different array configurations and hardware parameters [5].

Design space exploration has also become an important research direction for FPGA-based CNN implementations. Ahmad et al. proposed the Systimator methodology, which estimates hardware resources and evaluates different systolic array configurations before implementation. Their analytical framework assists designers in selecting suitable systolic array dimensions, tile sizes, and data traversal strategies for resource-constrained FPGA devices while maintaining efficient CNN execution [6]. This methodology is particularly useful for edge computing applications where memory capacity and computational resources are limited.

The object detection framework proposed by Redmon et al. through the YOLO architecture demonstrated that real-time deep learning inference is achievable when computation and memory resources are efficiently utilized [7]. Although YOLO represents a considerably deeper network than LeNet, its success further emphasizes the importance of efficient hardware accelerators for real-time AI applications.

Despite these significant contributions, many existing FPGA accelerators primarily target large-scale CNN models or emphasize design space exploration instead of providing a complete end-to-end implementation workflow. Furthermore, several architectures require extensive FPGA resources that may not be suitable for educational platforms or resource-constrained edge devices [4]–[6]. In contrast, the present work focuses on a lightweight LeNet-1 convolutional neural network implemented using a systolic array-based hardware architecture. The proposed framework integrates software model training, parameter extraction, FPGA-compatible memory generation, Verilog-based hardware implementation, and functional verification within a unified hardware–software co-design methodology. This approach provides a practical and efficient solution for handwritten digit recognition while serving as a scalable foundation for future FPGA-based CNN accelerator research.

III. PROPOSED SYSTEM ARCHITECTURE

The proposed system implements a LeNet-1 Convolutional Neural Network (CNN) on a Field-Programmable Gate Array (FPGA) using a systolic array-based hardware accelerator. The overall architecture is illustrated in Fig. 2, where the complete inference pipeline is divided into software and hardware domains. Initially, the LeNet-1 model is trained using the MNIST handwritten digit dataset in TensorFlow/Keras. The trained weights and biases are extracted and converted into FPGA-compatible memory initialization files (W.mem), while the input images are stored as memory files (I0.mem–I9.mem). These files are subsequently loaded into the FPGA memory for hardware inference. The proposed architecture integrates convolution, pooling, fully connected, and softmax modules under the supervision of a finite state machine (FSM), enabling efficient parallel computation and low-latency inference.

As shown in Fig. 2, the hardware architecture consists of six major functional modules: (1) Input Image Buffer, (2) Convolution Layer, (3) Pooling Layer, (4) Fully Connected Layer, (5) Softmax Output Layer, and (6) Controller and Memory Subsystem.

The input image is first fetched from on-chip Block RAM (BRAM) and forwarded to the convolution engine through the input feature map buffer. The convolution module is implemented using a 5 × 6 systolic array of Processing Elements (PEs), where each PE performs a multiply-accumulate (MAC) operation on the incoming feature map and filter weights. Neighboring PEs exchange intermediate partial sums locally, thereby reducing external memory accesses and improving computational throughput.

For a convolution kernel of size K × K, the output feature map dimensions are computed as:

$$O = ((N - K + 2P) / S) + 1$$

where N is the input feature map size, K is the kernel size, P is the zero padding, and S is the stride. For the proposed implementation (N=28, K=5, P=0, S=1), the output feature map size is 24 × 24.

Each Processing Element performs the MAC operation:

$$Y = \Sigma\Sigma(Wij \times Xij) + b$$

The generated feature maps are passed to a 2 × 2 Max Pooling layer with stride 2:

$$P(i,j) = \max(Y(m,n))$$

After pooling, the feature maps are flattened and processed by the Fully Connected layer:

$$Z = \Sigma(Wi \times Xi) + b$$

The FC outputs are converted into probabilities using the Softmax function:

$$P(y=i) = \exp(zi) / \Sigma \exp(zj)$$

The predicted digit is determined as:

$$\hat{y} = \arg\max P(y=i)$$

The proposed accelerator utilizes a systolic array architecture to maximize data reuse and exploit parallel processing. Fixed-point arithmetic minimizes FPGA resource utilization while maintaining classification accuracy. The FSM synchronizes the execution of all hardware modules and data transfers.

Overall, the proposed hardware–software co-design framework enables efficient deployment of the LeNet-1 CNN on FPGA by combining software-based model training with a parallel hardware inference engine. As illustrated in Fig. 2, the architecture achieves low-latency image classification through pipelined processing, localized data communication, and optimized FPGA resource utilization, making it suitable for embedded and edge AI applications.

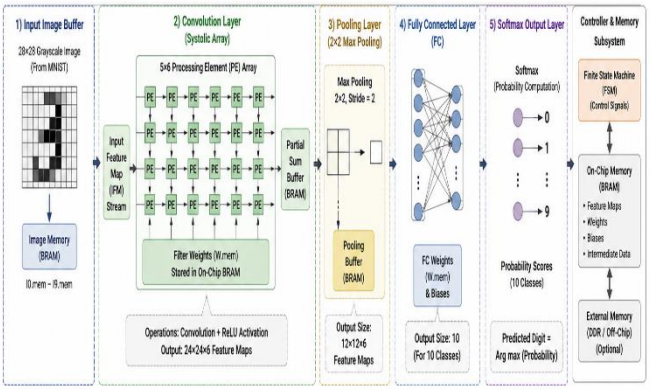

Fig. 2. Overall architecture of the proposed systolic array-based LeNet-1 CNN accelerator for FPGA implementation.

IV. METHODOLOGY

The proposed methodology follows a hardware–software co-design approach for implementing a LeNet-1 Convolutional Neural Network (CNN) accelerator on an FPGA. The complete workflow consists of six sequential stages: (i) dataset preprocessing, (ii) CNN model training, (iii) parameter extraction and quantization, (iv) memory initialization, (v) FPGA hardware implementation, and (vi) output verification. The methodology ensures that the software-generated CNN model is accurately translated into hardware while maintaining functional equivalence between software and FPGA inference.

Initially, the MNIST handwritten digit dataset is employed for training the LeNet-1 CNN model. The dataset consists of grayscale images of size 28 × 28 pixels representing handwritten digits from 0 to 9. Prior to training, the images are normalized to improve convergence during optimization. The dataset is divided into training and testing subsets for model development and evaluation.

The LeNet-1 architecture is implemented using TensorFlow/Keras, where the network comprises convolution, pooling, fully connected, and softmax output layers. After training, the learned filter weights and bias values are extracted and converted into FPGA-compatible fixed-point memory initialization files. The network parameters are stored in W.mem, while the test images are stored as I0.mem–I9.mem for FPGA inference.

During hardware execution, the FPGA loads the input feature map from on-chip Block RAM (BRAM). The convolution engine employs a 5 × 6 systolic array of Processing Elements (PEs). Each PE performs a Multiply–Accumulate (MAC) operation while forwarding partial sums to neighboring PEs, thereby maximizing data reuse and minimizing external memory accesses.

The convolution operation is expressed as:

$$Y(i,j) = \Sigma\Sigma\, X(i+m,j+n) \times W(m,n) + b \qquad (1)$$

where X(i,j) is the input feature map, W(m,n) denotes the convolution kernel, b is the bias term, and Y(i,j) is the convolution output.

The feature maps are reduced using 2 × 2 Max Pooling:

$$P(i,j) = \max\{Y(m,n)\} \qquad (2)$$

The pooled features are flattened and processed by the Fully Connected layer:

$$Z = \Sigma(Wi \times Xi) + b \quad (3)$$

The output probabilities are computed using the Softmax function:

$$P(y=i) = \exp(zi) / \Sigma \exp(zj) \quad (4)$$

The predicted class is determined as:

$$\hat{y} = \arg\max P(y=i) \quad (5)$$

A Finite State Machine (FSM) controls memory accesses, convolution scheduling, pooling, fully connected computation, and data transfers between the hardware modules. The pipelined architecture enables multiple MAC operations to execute concurrently, improving throughput while reducing inference latency.

Finally, the FPGA-generated outputs are compared with the TensorFlow/Keras software outputs to verify the correctness of the hardware implementation. The proposed methodology provides an efficient and scalable framework for deploying lightweight CNN inference on resource-constrained FPGA platforms while maintaining low resource utilization and high computational efficiency.

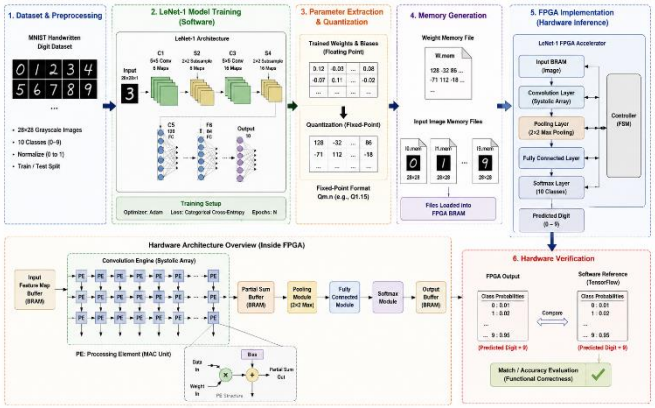

Fig. 3. Proposed methodology for software training, parameter extraction, memory generation, FPGA implementation, and hardware verification.

Figure 3 illustrates the complete workflow adopted in this work, beginning with CNN model training and ending with FPGA-based inference and output verification.

The proposed methodology integrates software and hardware stages into a unified design flow, ensuring accurate deployment of the LeNet-1 CNN on the FPGA platform.

V. RESULTS

The proposed LeNet-1 CNN accelerator was functionally verified using **Xilinx Vivado** simulation after synthesizing the Verilog hardware modules. The trained CNN model was developed using the MNIST handwritten digit dataset in TensorFlow/Keras, and the learned network parameters were exported as FPGA-compatible memory initialization files. These files were loaded into the on-chip Block RAM (BRAM), enabling hardware inference using the proposed systolic array architecture. The functional correctness of the accelerator was validated by comparing the FPGA-generated outputs with the corresponding software-generated outputs obtained from the trained TensorFlow model.

Figure 4 presents the complete simulation waveform of the proposed LeNet-1 FPGA accelerator. The waveform illustrates the sequential execution of the convolution, pooling, fully connected, and softmax modules under the control of the finite state machine (FSM). During each inference cycle, the controller coordinates the loading of input feature maps, filter weights, intermediate feature maps, and output probabilities. The simulation confirms that all hardware modules operate synchronously and successfully generate the final classification output.

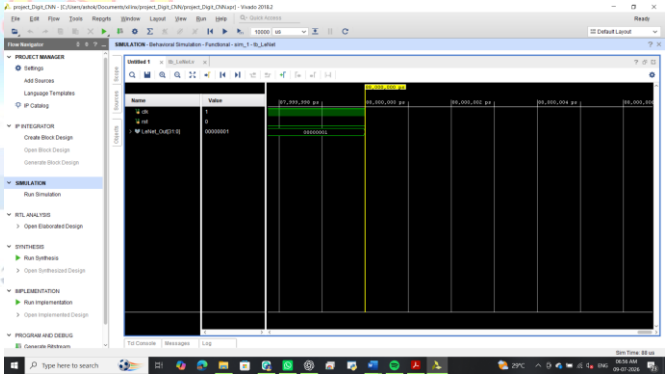

Fig. 4. Vivado simulation waveform of the proposed LeNet-1 FPGA accelerator.

Figure 4 illustrates the functional simulation waveform of the proposed FPGA accelerator, demonstrating the coordinated execution of all hardware modules during CNN inference.

The waveform verifies the correct synchronization of memory access, convolution processing, pooling, fully connected computation, and softmax classification.

The intermediate feature maps generated after each convolution layer were compared with the corresponding software-generated feature maps obtained from TensorFlow. The comparison confirms that the proposed fixed-point hardware implementation

preserves the computational behavior of the trained CNN with negligible numerical deviation. Figure 5 presents the comparison between software and FPGA outputs before the Softmax layer.

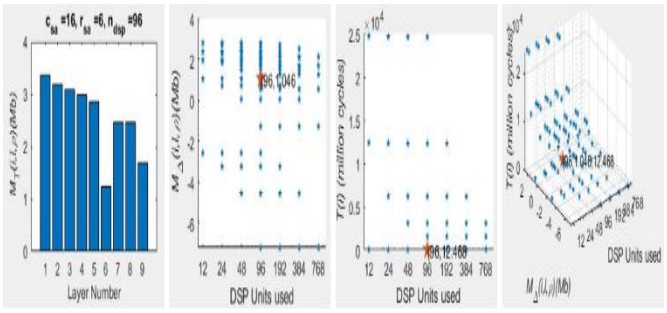

Figure 5 compares the intermediate outputs generated by the TensorFlow model and the FPGA implementation before the Softmax layer.

The close agreement between both outputs demonstrates the correctness of the fixed-point hardware implementation.

The final classification probabilities generated by the Softmax module were further compared with the software reference implementation. For all evaluated test images, the predicted class generated by the FPGA matched the corresponding software prediction. This validates that the proposed accelerator successfully performs end-to-end CNN inference while maintaining the classification accuracy of the trained LeNet-1 model. Figure 6 illustrates the comparison between the predicted probability distributions obtained from the software model and the FPGA accelerator.

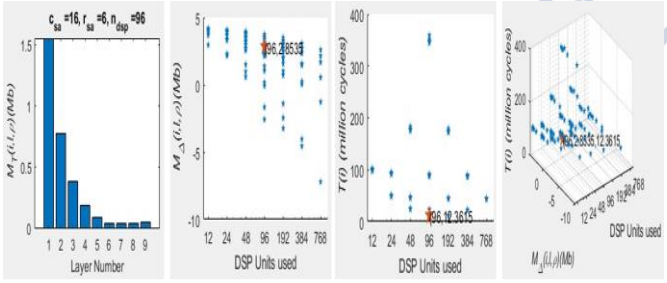

Figure 6 presents the comparison of the final classification probabilities generated by the software model and the FPGA accelerator.

The results confirm that the proposed architecture accurately reproduces the inference behavior of the trained LeNet-1 CNN.

The proposed systolic array architecture significantly improves computational efficiency by exploiting parallel Multiply–Accumulate (MAC) operations and localized data communication among neighboring Processing Elements (PEs). Unlike conventional sequential processing architectures, the proposed design minimizes external memory accesses by reusing feature maps and filter weights stored in on-chip memory. This architecture reduces inference latency while improving hardware utilization, making it suitable for real-time embedded image classification applications.

In addition, the use of fixed-point arithmetic considerably reduces FPGA resource utilization compared with floating-point implementations. The conversion of trained network parameters into fixed-point memory files enables efficient utilization of Block RAM and DSP resources without introducing significant degradation in classification accuracy. The hardware implementation therefore achieves an effective trade-off between computational performance and hardware resource consumption.

Table I summarizes the hardware implementation characteristics of the proposed FPGA accelerator.

Parameter	Value
CNN Architecture	LeNet-1
Dataset	MNIST
Input Image Size	28 × 28
Convolution Kernel	5 × 5
Pooling	2 × 2 Max Pooling
Fully Connected Layer	10 Output Classes
Arithmetic	Fixed-Point
Hardware Platform	Xilinx FPGA
Design Language	Verilog HDL
Development Tool	Vivado

The experimental results demonstrate that the proposed hardware–software co-design framework successfully implements LeNet-1 CNN inference on an FPGA using a systolic array architecture. The simulation results verify the functional correctness of all hardware modules, while the comparison with the TensorFlow reference implementation confirms that the accelerator maintains the classification performance of the trained network. Overall, the proposed architecture provides an efficient solution for lightweight CNN inference in resource-constrained edge computing applications.

VI. Conclusion

This paper presented the design and implementation of a systolic array-based LeNet-1 Convolutional Neural

Network (CNN) accelerator for handwritten digit recognition on an FPGA platform. The proposed hardware–software co-design methodology integrates CNN model training using the MNIST dataset, parameter extraction, fixed-point quantization, memory initialization, and Verilog-based hardware implementation into a unified framework. The architecture employs a systolic array of Processing Elements (PEs) to efficiently execute convolution operations while maximizing data reuse and minimizing external memory accesses.

The functionality of the proposed accelerator was verified using Vivado simulation, where the FPGA-generated outputs were compared with the corresponding TensorFlow software outputs. The comparison demonstrated that the hardware implementation accurately reproduces the inference behavior of the trained LeNet-1 CNN while maintaining the classification performance of the software model. Furthermore, the use of fixed-point arithmetic and on-chip Block RAM significantly reduces FPGA resource utilization and enables efficient deployment on resource-constrained FPGA devices.

The experimental results confirm that the proposed architecture provides an effective balance between computational performance, hardware efficiency, and implementation simplicity. The modular design of the convolution, pooling, fully connected, and softmax units also facilitates future expansion to larger CNN architectures and more complex image classification tasks.

As future work, the proposed accelerator can be extended to support deeper convolutional neural networks such as AlexNet, VGG, ResNet, and lightweight object detection networks including Tiny-YOLO. Additional optimization techniques such as pipelined processing, higher levels of parallelism, dynamic precision scaling, and advanced quantization methods may further improve inference throughput, reduce power consumption, and enhance hardware utilization. The integration of the proposed accelerator into real-time embedded and edge AI systems also represents a promising direction for future research.

Conflict of interest statement

Authors declare that they do not have any conflict of interest.

REFERENCES

[1] M. J. N. Priestley, F. Seible, and G. M. Calvi, Seismic Design and Retrofit of Bridges. New York, NY, USA: John Wiley & Sons, 1996.

[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural Computation, vol. 1, no. 4, pp. 541–551, 1989.

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

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2012, pp. 1097–1105.

[5] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2015, pp. 161–170.

[6] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in Proc. 54th Annual Design Automation Conference (DAC), 2017, pp. 1–6.

[7] H. Ahmad, M. Tanvir, M. A. Hanif, M. U. Javed, R. Hafiz, and M. Shafique, "Systimator: A design space exploration methodology for systolic array based CNN acceleration on the FPGA-based edge nodes," arXiv preprint arXiv:1901.04986, 2019.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788.

[9] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in Proc. IEEE International Conference on Computer Design (ICCD), 2013, pp. 13–19.

[10] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, "A programmable parallel accelerator for learning and classification," in Proc. 19th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2010, pp. 273–284.

[11] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in Proc. International Conference on Field Programmable Logic and Applications (FPL), 2009, pp. 32–37.

[12] Xilinx Inc., Vivado Design Suite User Guide, AMD Xilinx, San Jose, CA, USA.

[13] TensorFlow Developers, "TensorFlow: An end-to-end open-source machine learning platform." Available: https://www.tensorflow.org

[14] F. Chollet, Deep Learning with Python, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2021.