

Implementation and Optimization of LeNet-5 Model for Handwritten Digits Recognition on FPGAs using Brevitas and FINN

Josphat Chege Njuguna
Electronics and Communication Engineering
Kocaeli University
Kocaeli, Turkey
chege.njuguna91@gmail.com

Aysun Taşyapı Çelebi
Electronics and Communication Engineering
Kocaeli University
Kocaeli, Turkey
aysun.tasyapi@kocaeli.edu.tr

Anıl Çelebi
Electronics and Communication Engineering
Kocaeli University
Kocaeli, Turkey
anilcelebi@kocaeli.edu.tr

Abstract— Handwritten digit recognition using deep learning models is an active area of research with wide-ranging applications. However, the high computational requirements of these models limit their deployment on resource-constrained devices. To address this challenge, we propose an optimized implementation of the LeNet-5 model for handwritten digit recognition on FPGAs using the Brevitas and FINN frameworks. We trained the model on the MNIST dataset and quantized it to reduce its memory footprint and improve inference performance. We then used FINN to generate hardware operators and RTL code, which were synthesized for an EFINIX FPGA. Additionally, we used a RISC-V processor and custom HLS VDMAs to interface with the FPGA and enable image loading and result display through GUI. Our approach achieved a classification accuracy of 96.64% on the test set, while consuming significantly lower power than CPU and GPU implementations. Our methodology demonstrates the potential of leveraging FPGA hardware acceleration to enable efficient deployment of deep learning models on resource-constrained devices.

Keywords—FPGA, BNN, LeNet-5, HLS, Brevitas, FINN, RISC-V, Quantization, Hardware Acceleration, CNN

I. INTRODUCTION

Handwritten digits recognition is a classic problem in computer vision that has been widely studied in literature. The LeNet-5 model, proposed by [1], is a well-known convolutional neural network (CNN) architecture that has been successfully used for handwritten digits recognition. In recent years, there has been growing interest in deploying CNNs on field-programmable gate arrays (FPGAs) to achieve high performance and energy efficiency. This literature review summarizes recent research on implementing and optimizing deep learning networks on FPGAs using the Brevitas and FINN frameworks.

Brevitas is a PyTorch-based framework that enables quantization-aware training (QAT) of deep neural networks (DNNs). Brevitas supports various types of quantization, including binary, ternary, and fixed-point quantization, which can be used to reduce the memory footprint and computational complexity of DNNs. In [2], presents an evaluation of the impact of mixed precision on neural networks deployed on

FPGAs. With the growing cost of both training and inference for state-of-the-art neural networks, the literature has looked for ways to reduce resources used with a minimal impact on accuracy. Deploying a network must be possible on low-power and low-resource hardware architectures. Reconfigurable architectures, such as FPGAs, have proven to be more powerful and flexible than GPUs when looking at a specific application. The authors use FINN and Brevitas, two frameworks from Xilinx labs, to assess the impact of quantization on neural networks using 2-to-8-bit precisions and weights with several parallelization configurations. They found that equivalent accuracy can be obtained using lower-precision representation and enough training. However, the compressed network can be better parallelized, allowing the deployed network throughput to be 62 times faster.

Additionally, [3] addressed the challenge of deploying quantized neural networks (QNNs) on heterogeneous all-programmable devices (APDs). These devices have different architectures and computation capabilities, making it challenging to optimize and deploy QNNs across them. The authors proposed a memory-efficient dataflow inference method for deep convolutional neural networks (CNNs) that could be executed on FPGAs. The method utilized parallelism and data reuse to optimize the computation, reducing the memory requirements for inference. The researchers evaluated their method on two FPGA-based platforms and demonstrated its effectiveness in reducing the memory footprint of CNNs, achieving high inference performance, and enabling the deployment of QNNs on heterogeneous APDs.

Initially, FINN utilized HLS library which had its own advantages and disadvantages. However, [4] presents an alternative backend library for FINN that leverages RTL instead of HLS for FPGA-based accelerators used in deep neural network inference. The authors investigate and compare the pros and cons of RTL-based implementation versus the original HLS variant. They show that for smaller design parameters, RTL produces significantly smaller circuits than HLS, while for larger circuits, the look-up table count of RTL-based design is slightly higher, up to around 15%. HLS

consistently requires more flip-flops and block RAMs, which also impacts the critical path delay, with RTL producing significantly faster circuits, up to around 80%. Moreover, RTL also benefits from at least a 10 \times reduction in synthesis time. The results were practically validated using a real-world use case of a multi-layer perceptron (MLP) network used in network intrusion detection. Overall, the authors suggest that the ease in the design entry is less important for HLS frameworks since they code-generate the hardware design, and the RTL abstraction might be an attractive alternative given the gained benefits in synthesis time together with some design-dependent resource benefits.

To further optimize the performance of DNN on FPGAs, [5] proposes a methodology for improving the on-chip memory utilization efficiency of custom dataflow convolutional neural network (CNN) inference accelerators. The proposed methodology, called Frequency Compensated Memory Packing (FCMP), is demonstrated on medium-sized CIFAR-10 inference accelerators and a quantized ResNet-50 CNN accelerator. The authors show that FCMP can reduce On-Chip Memory (OCM) utilization by up to 30% without loss of inference throughput, allowing for portability to smaller FPGA boards. The proposed methodology also provides a finer-grained trade-off between throughput and OCM requirements, increasing the flexibility of custom dataflow CNN inference designs on FPGA. In addition, [6] investigates the trade-off between accuracy and throughput for reduced-precision neural networks on reconfigurable logic. The authors propose a quantization training strategy that allows for competitive model accuracy with a lower memory footprint, and their experiments show that 2-bit and 4-bit fixed-point parameters provide the best trade-off on small-scale datasets, while 4-bit parameters show the best trade-off on large-scale tasks.

The applications of FINN in real-time scenarios have caught the attention of many researchers across diverse fields. For instance, object detection systems based on deep learning have become crucial in numerous applications such as robotics and the automotive industry. FINN has enabled significant achievements in addressing the challenge of deploying deep learning models on resource-constrained and energy-first devices. A YOLO-like object detection algorithm was proposed in [7], which was deployed on an FPGA platform to take advantage of the FPGA's computational features. The researchers were able to accelerate the computational units in the model, such as convolution, pooling, and concatenation layers, for inference by developing corresponding hardware operators based on the model units. The proposed object detection accelerator was implemented and verified on the Xilinx ZYNQ platform. The experimental results showed that the detection accuracy of the algorithm model was comparable to that of common algorithms, while the power consumption was much lower than that of the CPU and GPU. The accelerator had a fast inference speed and was suitable for deployment on mobile devices for detecting the surrounding environment.

Furthermore, [8] proposed an FPGA implementation of the low precision YOLO (LPYOLO) architecture for face detection. Taking advantage of Brevitas and FINN platform, the authors used fixed-point quantization and FPGA acceleration to achieve real-time performance with low power consumption on an FPGA. The authors also compared the performance of LPYOLO to other state-of-the-art face detection algorithms and showed that LPYOLO achieved comparable accuracy with much lower energy consumption. Finally, [9] used the FINN framework to generate FPGA-accelerated neural network designs with reduced bit precision through quantization. This allowed for improved hardware efficiency while maintaining high accuracy on various datasets, including the MNIST dataset used in their experiments. The authors also incorporated knowledge distillation, an approach that transfers knowledge from a larger teacher network to a smaller student network, to further improve the performance of their quantized networks. Overall, their work demonstrates the potential of using quantization and knowledge distillation with FPGA-accelerated neural networks for efficient and high-performing hardware implementations. The utilization of FINN compiler in the three instances results in a reduced development and deployment time, as a significant portion of the work is taken care of by the compiler. This enables the products to be market-ready sooner.

There are alternative approaches for deploying DNNs on FPGAs besides using FINN and Brevitas. Some methods require writing code in Verilog or VHDL, while others utilize HLS for model inference. For example, a study by [10] presents an FPGA-based implementation of LeNet-5 CNN architecture using HLS. The authors have achieved a high level of parallelism and optimized the hardware resources by using a pipelined architecture, loop unrolling, and memory buffering. However, their approach requires manual tuning of design parameters, which can be time-consuming and challenging. In comparison, the FINN framework uses automated optimizations and graph-based transformations to streamline the design process and achieve high performance with minimal manual intervention.

Moreover, [11] proposed an improvement on the traditional LeNet-5 architecture by leveraging FPGA hardware acceleration. Their approach achieved higher efficiency and accuracy compared to the original LeNet-5 model. The authors trained their improved LeNet-5 model on handwritten digit recognition tasks and showed promising results. Notably, their work highlights the potential of using FPGA hardware to enhance the performance of deep learning models. However, the author did not provide more details on FPGA resource utilization that could have been of great importance to compare with our study.

Overall, these studies highlight the potential of FPGAs for accelerating the inference of deep neural networks, especially for resource-constrained and energy-efficient devices. They demonstrate the importance of optimizing the neural network models for the FPGA architecture, including quantization and

hardware operator implementation. The process of implementing hardware operators based on the neural network model units is a crucial step towards achieving efficient deployment of deep learning models on FPGA platforms. Various studies have proposed different approaches to implement hardware operators, including hand-crafted RTL designs, automated code generation tools, and HLS-based designs.

II. HARDWARE ARCHITECTURE

A. FINN Design Flow

FINN is an end-to-end flow for building fast and efficient FPGA implementations of neural networks. It provides a set of high-level abstractions for defining, optimizing, and implementing neural network models on FPGAs. The flow starts with a trained neural network model, and then optimizes and quantizes it for implementation on FPGAs. FINN also generates FPGA-specific accelerator code in RTL form, which can be synthesized and deployed on the target FPGA. FINN supports a wide range of neural network models and can generate efficient FPGA implementations for them. To use FINN, the user first needs to define and train a neural network using a deep learning framework such as PyTorch or TensorFlow. Once the network is trained, it can be exported to FINN, where it undergoes optimization and quantization for implementation on the FPGA. FINN then generates FPGA-specific accelerator code and RTL for the network, which can be synthesized and deployed on the target FPGA [12].

B. LeNet-5 Design Flow

The overall design flow for the LeNet-5 implementation is depicted in Fig. 1. First, the network is quantized and trained in Google Colab, achieving an accuracy of 96.64%. The trained model is then exported as an ONNX (Open Neural Network Exchange) module and imported into the FINN framework. FINN applies graph-based transformations and optimizations to the ONNX model to streamline and partition it into FPGA-friendly components, which are then mapped onto the

Efinix FPGA. The CNN architecture is created with quantized layers and activation functions of 3 integer bit width precision. The weights are signed integers, and the activations are unsigned integers. 8-bit integer values are preferred in the first and last convolution layers for better accuracy. Quantized layers and activation functions are provided by Brevitas library in PyTorch. In the ONNX model optimization phase, folding and configuration parameters are set to provide lower latency. Folding parameters determine how many Processing Elements (PE) and Single Instruction Multiple Data (SIMD) will be used on each layer. Increasing the number of PE and SIMD boosts the parallelism and produces higher speed, but it consumes more logical resources. Configuration parameters decide the type of memory usage, such as BRAM or LUTRAM of the FPGA for each layer. The type of logical resource usage, such as LUT or DSP of the FPGA, is also set for each layer. Intellectual Property (IP) blocks are created for each layer from the ONNX model with Xilinx Vitis HLS tool using FINN-HLS library, which contains layer definitions of the created model in C++. The created IP blocks are stitched together and synthesized via Efinity tool.

C. LeNet-5 Inference Flow

The Efinix Trion FPGA platform was chosen for its low power consumption and small form factor, and the RISC-V processor was utilized to interface with the GUI, offering benefits such as modularity, extensibility, reduced licensing costs, increased flexibility, and improved transparency. RISC-V's open-source nature also enables collaboration and innovation in the development of new hardware and software solutions, making it an increasingly popular choice for FPGA-based systems. As shown in Fig. 2, the implementation was tested using a GUI to load a 28×28 image via UART to RISC-V, which sends the image data to DDR memory and utilizes a custom HLS VDMA to read the image into the LeNet-5 accelerator. The results are then sent back to RISC-V using AXI Lite and displayed over the GUI. The output is then compared with the input in real-time as shown in Fig 2.

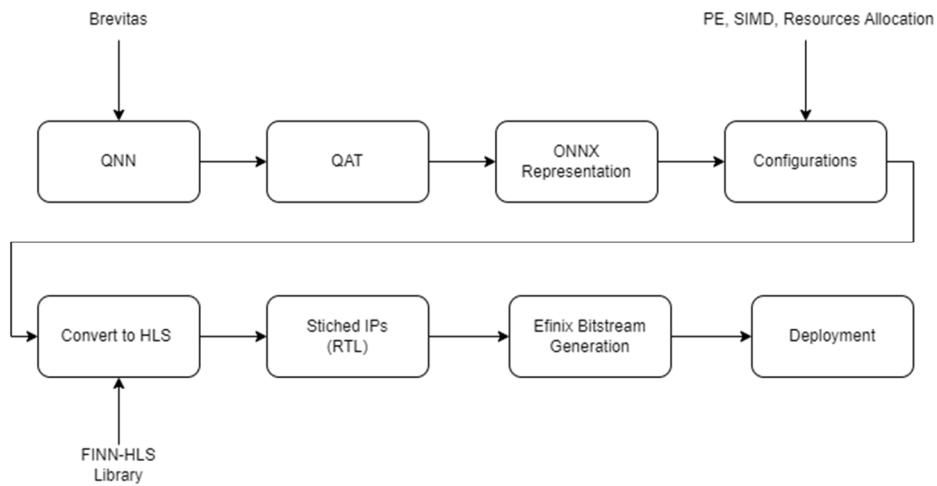


Fig. 1. LeNet-5 Design Flow

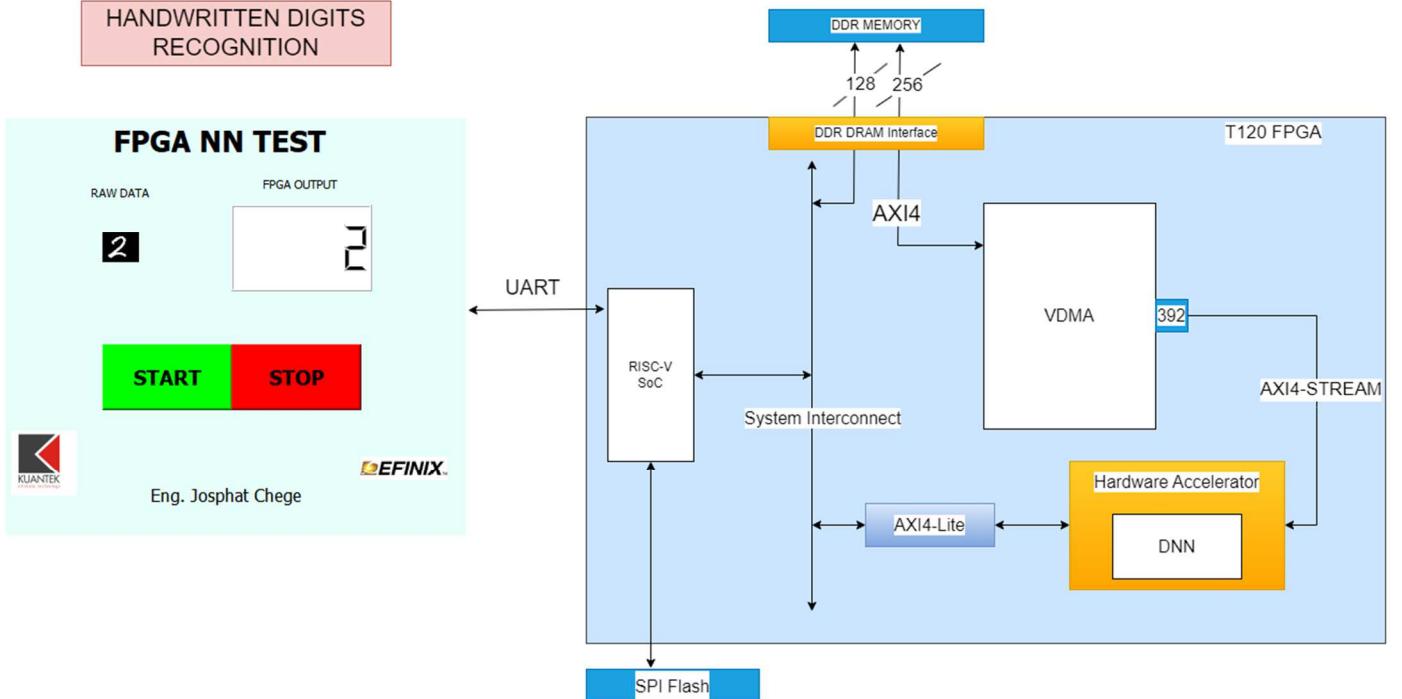


Fig. 2. LeNet-5 Inference Flow Block Design.

III. EXPERIMENTAL RESULTS

The LeNet-5 implementation was assessed using the MNIST dataset and achieved an accuracy of 96.64%. The system was implemented on the Efinix Trion FPGA platform, and its maximum operating frequency was 100 MHz, with an average power consumption of 0.62 W. A 10x speedup was observed when compared to a software implementation running on a CPU. The resource utilization is shown in TABLE I. In [10], a similar LeNet-5 implementation achieved 4.6x speedup, with 125 DSPs, 119.5 BRAMs, 14659 LUT, 14172 flipflops, and a power consumption of 1.8 W. Additionally, TABLE I include comparison of resource utilization between [10] and our research. However, the proposed system in this study utilized only 4 DSPs due to the quantization of weights and biases, and the power consumption was significantly lower. FINN enables the storage of weights and biases on on-chip RAM, which reduces DDR access, power consumption, and latency.

TABLE I.

| Block | Flip Flops | LUTs | BRAMs | DSPs |
|------------|------------|-------|-------|------|
| This paper | 10359 | 9571 | 101 | |
| [10] | 14172 | 14659 | 119.5 | 125 |

IV. CONCLUSION

In this work, we presented a methodology for implementing and optimizing the LeNet-5 model for handwritten digit recognition on FPGAs using Brevitas and FINN. The proposed system was implemented on the Efinix Trion FPGA platform and demonstrated high accuracy, low power consumption, and

improved throughput compared to a software implementation. The use of a custom RISC-V processor and VDMAs provided efficient data transfer and improved overall system performance. The proposed Python GUI also provided a convenient and user-friendly interface for testing and evaluation. Brevitas is a PyTorch-based quantization-aware training and validation framework that allows users to train deep neural networks with low bit-width weights and activations. It provides a set of predefined quantization schemes and can also be extended to support user-defined schemes. Brevitas allows users to easily define and train networks with integer weights and activations, which can then be exported for implementation on an FPGA using the FINN toolchain.

Overall, Brevitas and FINN make it easier for developers to implement neural networks on FPGAs, by providing high-level abstractions and tools that automate the process of optimization and implementation.

ACKNOWLEDGMENT

The authors express their gratitude to KuanTek Elektronik A.S. for providing the necessary hardware and software infrastructure to conduct this research.

REFERENCES

- [1] Y. Lecun, L. Eon Bottou, Y. Bengio, and P. H. Abstract, "Gradient-Based Learning Applied to Document Recognition," 1998.
- [2] Ducasse Q, Cotret P, Lagadec L, and Stewart R, "Benchmarking Quantized Neural Networks on FPGAs

- with FINN,” 2021. [Online]. Available: <https://github.com/QDucasse/>
- [3] M. Blott *et al.*, “FinN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks,” *ACM Trans Reconfigurable Technol Syst*, vol. 11, no. 3, Dec. 2018, doi: 10.1145/3242897.
- [4] S. A. Alam, D. Gregg, G. Gambardella, T. Preusser, and M. Blott, “On the RTL Implementation of FINN Matrix Vector Compute Unit,” Jan. 2022, [Online]. Available: <http://arxiv.org/abs/2201.11409>
- [5] L. Petrica, T. Alonso, M. Kroes, N. Fraser, S. Cotofana, and M. Blott, “Memory-Efficient Dataflow Inference for Deep CNNs on FPGA,” Nov. 2020, [Online]. Available: <http://arxiv.org/abs/2011.07317>
- [6] J. Su *et al.*, “Accuracy to Throughput Trade-offs for Reduced Precision Neural Networks on Reconfigurable Logic,” Jul. 2018, [Online]. Available: <http://arxiv.org/abs/1807.10577>
- [7] X. Zheng and T. He, “Reduced-Parameter YOLO-like Object Detector Oriented to Resource-Constrained Platform,” *Sensors (Basel)*, vol. 23, no. 7, Apr. 2023, doi: 10.3390/s23073510.
- [8] B. Gunay, S. B. Okcu, and H. S. Bilge, “LPYOLO: Low Precision YOLO for Face Detection on FPGA,” in *Proceedings of the World Congress on Electrical Engineering and Computer Systems and Science*, Avestia Publishing, 2022. doi: 10.11159/mvml22.108.
- [9] R. Stewart, A. Nowlan, P. Bacchus, Q. Ducasse, and E. Komendantskaya, “Optimising hardware accelerated neural networks with quantisation and a knowledge distillation evolutionary algorithm,” *Electronics (Switzerland)*, vol. 10, no. 4, pp. 1–21, Feb. 2021, doi: 10.3390/electronics10040396.
- [10] Rongshi D and Yongming T, *Accelerator Implementation of Lenet-5 Convolution Neural Network Based on FPGA with HLS*. 2019.
- [11] Y. Hou and Z. B. Chen, “LeNet-5 improvement based on FPGA acceleration,” *The Journal of Engineering*, vol. 2020, no. 13, pp. 526–528, Jul. 2020, doi: 10.1049/joe.2019.1190.
- [12] Y. Umuroglu *et al.*, “FINN: A Framework for Fast, Scalable Binarized Neural Network Inference,” Dec. 2016, doi: 10.1145/3020078.3021744.