JOURNAL OF ELECTRONIC VOLTAGE AND APPLICATION
e-ISSN: 2716-6074

JEVA

Vol. 5 No. 1 (2024) 35-41
https://publisher.uthm.edu.my/ojs/index.php/jeva

Hardware Implementation of the Activation Layer and Mean Pooling Layer for the CNN Digit Recognition

Chessda Uttraphan[1], Araveindran Sithamparam[1]

[1] *Faculty of Electrical and Electronic Engineering,*
University Tun Hussein Onn Malaysia, Batu Pahat, Johor, 86400, MALAYSIA

*Corresponding Author: chessda@uthm.edu.my
DOI: https://doi.org/10.30880/jeva.2024.05.01.004

Article Info

Received: 5 January 2024
Accepted: 17 June 2024
Available online: 30 June 2024

Keywords

Deep learning, CNN digit recognition, hardware CNN, pooling layes, FPGA, hardware accelerator

Abstract

This work focuses on enhancing the efficiency of Convolutional Neural Networks (CNNs) for digit recognition through dedicated hardware design of the ReLU activation function and mean pooling layer. The CNN model is initially implemented in MATLAB and trained on the MNIST dataset. The hardware architecture, designed using Verilog HDL for an Intel Cyclone IV E FPGA, successfully replicates the MATLAB outputs, as verified through rigorous simulations with ModelSim. The hardware implementation demonstrates significant performance improvements, notably reducing execution time from 104,458µs in software to 8.05µs in hardware. The designed hardware exhibits a 2.60GHz frequency, 4,457 logic elements, 2,522 registers, 409,600 memory bits, and 71.73mW thermal power dissipation, showcasing superior computational efficiency.

1. Introduction

In the realm of image processing, computer vision and pattern recognition are a key emerging field. One of the key areas in pattern recognition is the handwritten digit recognition system [1]. Machine learning is all about developing and implementing algorithms that make these conclusions and predictions easier to recognize and less time-consuming [2]. In recent years, one of the most important tasks has been Handwritten Digit Recognition for various purposes, including bank checking of handwritten digits and conversion into machine-readable formats, vehicle number plate recognition for efficient monitoring and identification, NID (National Identification) verification, etc [3]. In the previous two decades, research has been ongoing in deep learning. Deep learning algorithms like CNN are broadly used for recognition [4]. CNN architecture is proposed to achieve accuracy even better than that of ensemble architectures, along with reduced operational complexity and cost [5]. A specialized artificial intelligence technique primarily used for image classification is the Convolutional Neural Network (CNN) [6]. CNNs, a subset of deep learning, are widely utilized for image categorization due to their processing efficiency and accuracy. The convolution and pooling layers may consist of more than one and transmit the data to the fully connected layer [7]. FPGA offers reprogramming ability and, when its hardware architecture is well designed, provides adequate energy efficiency. FPGA can satisfy power and size constraints, emerging as a design alternative. It has both pipeline parallelism and data parallelism, so it provides lower latency for processing tasks, becoming a high-performance and flexible accelerator for CNN inference [8]. The Modified National Institute of Standards and Technology (MNIST) handwritten digit database, one of the most important areas of research in pattern recognition, has excellent research and practical value [9]. The study specifically addresses the challenges of software implementation, such as slow computation with a single arithmetic logic unit (ALU) and high-power consumption for CNNs with numerous nodes [10].

The work outlines objectives, including creating an optimized hardware architecture, verification through software comparison, and benchmarking performance metrics. The scope encompasses the implementation of the CNN model in MATLAB, focusing on the ReLU and mean pooling layers, recognizing digits zero to nine in

28×28-pixel images. The software implementation will be in MATLAB, while the hardware design will use Intel Quartus Prime with Verilog HDL. Simulation using ModelSim will validate hardware results against MATLAB outputs, emphasizing speed, logic elements, and power consumption as performance metrics. The report concludes with the expectation of improved computational efficiency in the proposed hardware design compared to the software implementation.

2. Methodology

Fig. 1 shows the overall work for the duration of this work. There are two phases; where phase one is focused on designing the CNN digit recognition in MATLAB, and Phase two focuses on the hardware implementation of digit recognition using FPGA.

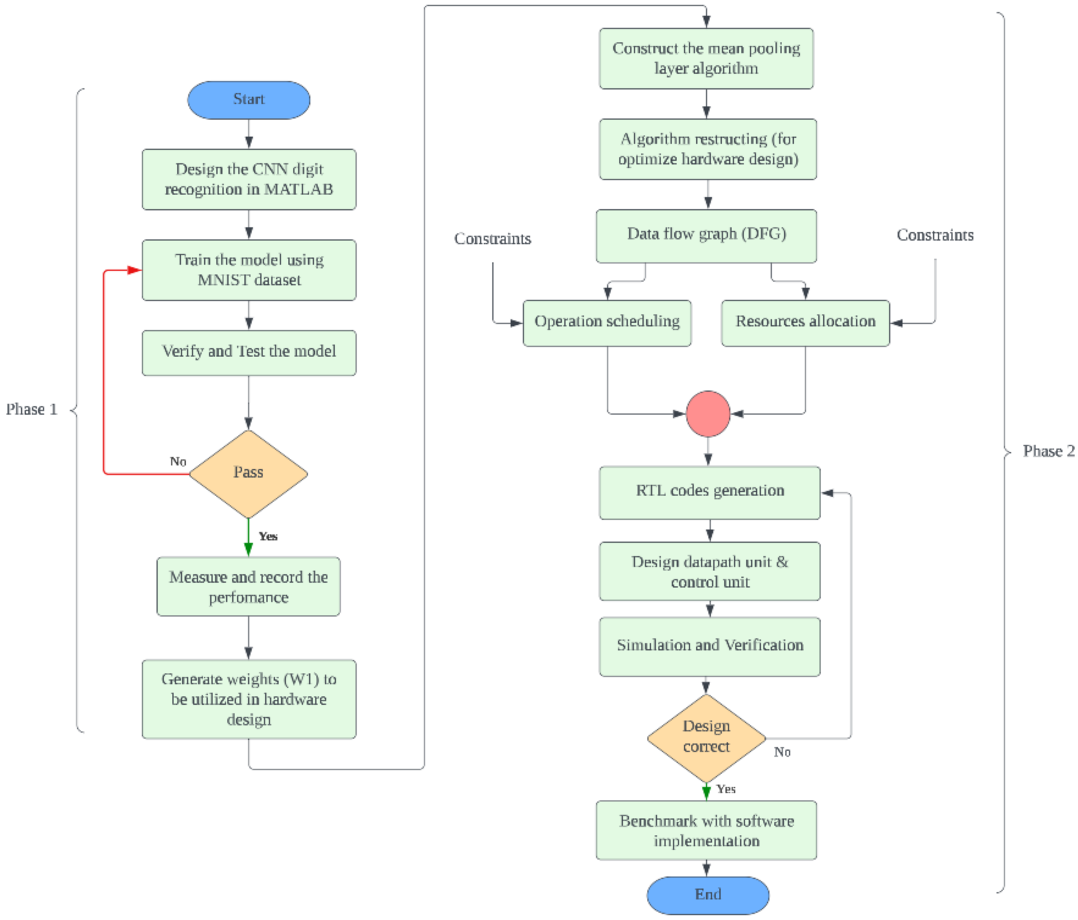

Fig. 1 *Methodology*

2.1 Designing the CNN Digit Recognition in MATLAB (Phase 1)

Table 1 *CNN digit recognition model summary*

Layer	Remark	Activation Function
Input	28×28 nodes	-
Convolution	20 convolution filter (9×9)	ReLU
Polling	1 mean pooling (2×2)	-
Hidden	100 nodes	ReLU
Output	10 nodes	Softmax

J. of Electronic Voltage and Application Vol. 5 No. 1 (2024) p. 35-41

In this section, a CNN model based on Table 1 is designed using MATLAB. In this model, it has 28×28 nodes in the Input layer, 20 convolution filters (9×9) in the Convolution layer, ReLU activation, one mean pooling (2×2) in the pooling layer, 100 nodes with ReLU activation in the Hidden layer, and 10 nodes with SoftMax activation in the Output layer. The model of CNN digit recognition is shown in Fig. 2.

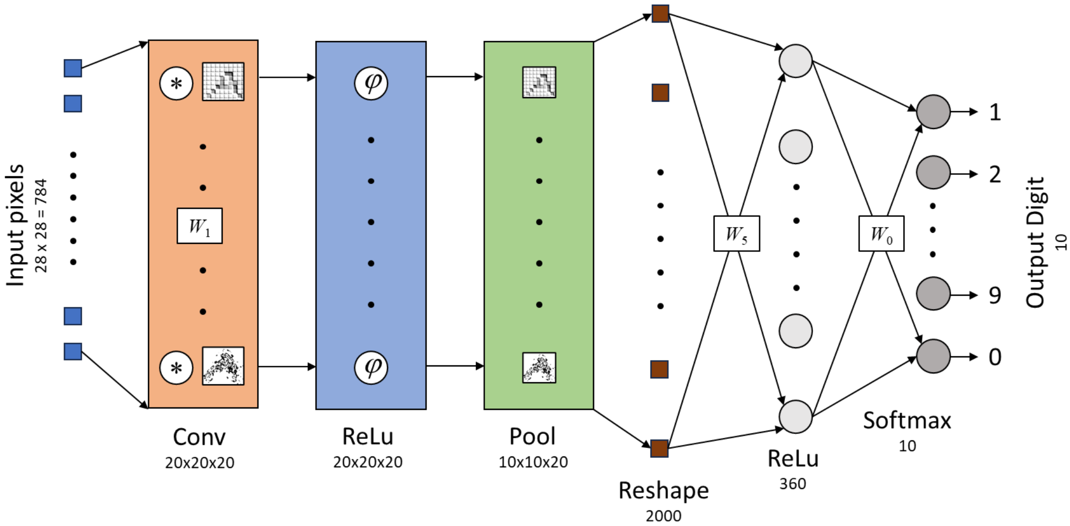

Fig. 2 *CNN digit recognition model*

The input will be from the MNIST dataset. The MNIST dataset comprises 60,000 images of handwritten digits (all grouped into one file. Each image is gray-scaled and positioned in a fixed size of 28 * 28 pixels, representing a number shown in Fig. 3. MATLAB, known as software implementation, extracts data at different stages of the CNN. Fig. 4 shows the overall operation in MATLAB.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.329412	0.72549	0.623529	0.592157	0.235294	0.141176	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.870588	0.996078	0.996078	0.996078	0.996078	0.945098	0.776471	0.776471	0.776471	0.776471	0.776471	0.776471	0.776471	0.776471	0.666667	0.203922	0	0	0	0	0	0
0	0	0	0	0	0	0.262745	0.447059	0.282353	0.447059	0.639216	0.890196	0.996078	0.882353	0.996078	0.996078	0.996078	0.980392	0.898039	0.996078	0.996078	0.54902	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.066667	0.258824	0.054902	0.262745	0.262745	0.262745	0.231373	0.082353	0.92549	0.996078	0.415686	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.32549	0.992157	0.819608	0.070588	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.086275	0.913725	1	0.32549	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.505882	0.996078	0.933333	0.172549	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.231373	0.976471	0.996078	0.243137	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.521569	0.996078	0.733333	0.019608	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.035294	0.803922	0.972549	0.227451	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.494118	0.996078	0.713725	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.294118	0.984314	0.941176	0.223529	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.07451	0.866667	0.996078	0.65098	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.011765	0.796078	0.996078	0.858824	0.137255	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.14902	0.996078	0.996078	0.301961	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.121569	0.878431	0.996078	0.45098	0.003922	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.521569	0.996078	0.996078	0.203922	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.239216	0.94902	0.996078	0.996078	0.203922	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.47451	0.996078	0.996078	0.858824	0.156863	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.47451	0.996078	0.811765	0.070588	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 3 *28 * 28 pixels image*

The data from each feature is important because it will be used to verify the hardware implementation functionality. The output data (Convolution + ReLu + Mean Pooling) from the MTALAB and hardware simulation must be consistent.

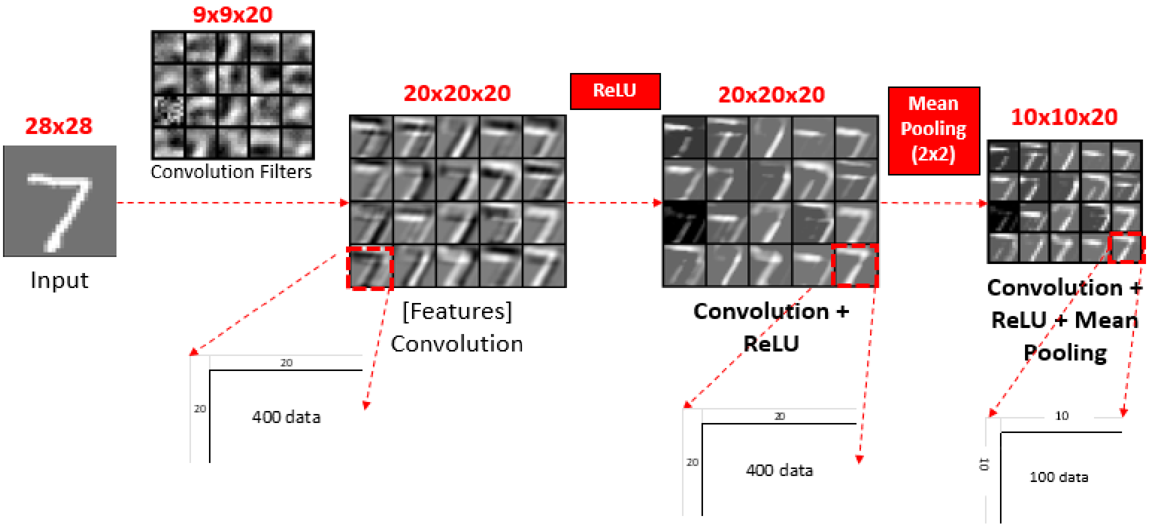

Fig. 4 *MATLAB overall operation*

2.2 Designing the Hardware Architecture

The overall design of the Hardware Implementation is shown in Fig. 5. The hardware design comprises 20 embedded memories for storing the inputs (Feature1 to Feature20), 20 for storing the outputs (Pool1 to Pool20), and ReLU and mean pooling block.

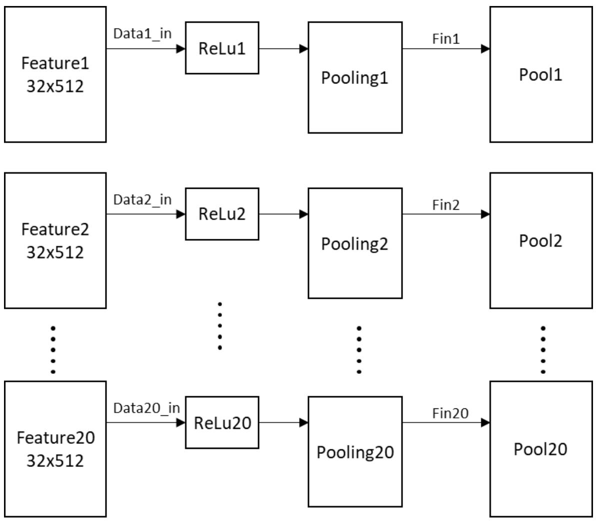

Fig. 5 *Top model architecture design*

Fig. 6 shows the design of ReLu and mean pooling. The model will check for the most significant bit. If the data is negative, then the data will be zero else the data passes input through. The Pooling model calculates the average of the values in registers R0 to R3. Summing the pairs of registers (R0+R1 and R2+R3). Adding those sums together then Performing a right shift by two, effectively dividing the sum by four to get the average. The mean pooling model is designed based on Equation 1.

$$Mean\ Pooling = \frac{(R0 + R1) + (R2 + R3)}{4} \quad \text{(Eq1)}$$

J. of Electronic Voltage and Application Vol. 5 No. 1 (2024) p. 35-41

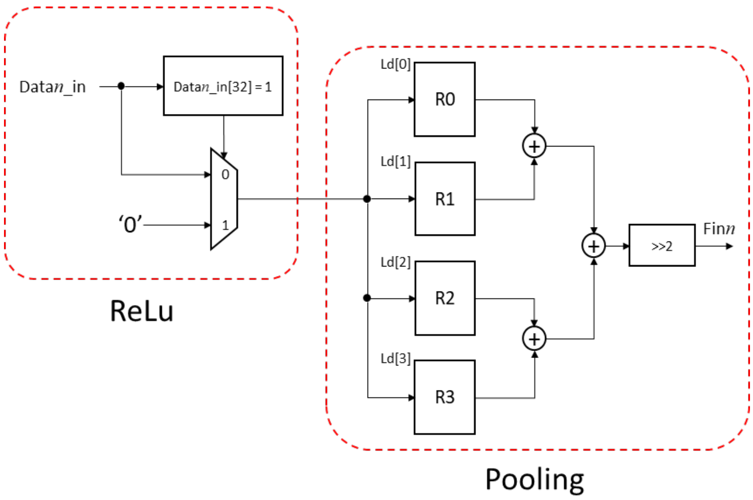

Fig. 6 *ReLU and mean pooling architecture*

3. Result and Discussion

To verify the hardware designed, the output data from MATLAB is compared with the output data in the output Memory file (Pool1 to Pool20). Both output data must be consistent as shown in Fig. 7 and Fig. 8.

Memory Data - /MeanPool_TB/DUT/RAM1/altsyncram_component/m_default/altsy

0	00000000	00000000	00000000	00000000	00000000
5	00000000	00000000	00000000	00000000	00000092
10	00001351	00000e8a	000005ab	00000340	00000034
15	00000000	00000000	00000000	00000000	00000179
20	00003482	00003c89	00003e56	00003c48	00003c77
25	0000297f	000012b5	000001ff	00000287	0000071a
30	000010d5	00000dc6	00000ab2	000007eb	00000a7f
35	00000546	00000a29	00000000	00000000	00000099
40	000003de	00000117	00000000	00000000	00000000
45	00000000	00000000	00000000	00000000	00000000
50	00000000	00000006	0000007c	00000000	000000ce
55	00001486	00000c06	00000000	00000000	00000000
60	00000000	00000000	00000000	00000000	0000080e
65	000012d6	00000313	00000000	00000000	00000000
70	00000000	00000000	00000000	0000028b	000018fd
75	00000f2d	00000000	00000000	00000000	00000000
80	00000000	00000000	00000000	0000055a	00000eae
85	000002c9	00000000	00000000	00000000	00000000
90	00000000	00000318	0000143d	00001487	000002b2
95	00000000	00000000	00000000	00000000	00000000
100	00000000	00000000	00000000	00000000	00000000
105	00000000	00000000	00000000	00000000	00000000
110	00000000	00000000	00000000	00000000	00000000
115	00000000	00000000	00000000	00000000	00000000
120	00000000	00000000	00000000	00000000	00000000
125	00000000	00000000	00000000		

Fig. 7 *Output stored in embedded memory (Hardware)*

0	0	0	0	0	0
5	0	0	0	0	92
10	1351	E8A	5AB	340	35
15	0	0	0	0	179
20	3482	3C89	3E56	3C48	3C77
25	2980	12B5	1FF	288	71B
30	10D6	DC6	AB3	7EC	A7F
35	547	A29	0	0	9A
40	3DE	117	0	0	0
45	0	0	0	0	0
50	0	7	7C	0	CF
55	1486	C06	0	0	0
60	0	0	0	0	80E
65	12D6	313	0	0	0
70	0	0	0	28B	18FD
75	F2D	0	0	0	0
80	0	0	0	55A	EAE
85	2C9	0	0	0	0
90	0	318	143D	1488	2B2
95	0	0	0	0	0

Fig. 8 *Output from MATLAB*

3.1 Performance Comparison

The performances of the software and the hardware are recorded. The software implementation's maximum operating frequency is determined by the capabilities of the device used for its execution. The speed for the hardware implementation is taken from the Quartus stimulation as shown in Fig. 9. The other performances metrics are taken from the Analysis & Synthesis Summary in Quartus. In Table 2 shows the comparison between the software and the hardware implementation.

Based on Table 2, the execution in software environment is 104458 µs while for the hardware it is only 8.05 µs. The hardware executes the algorithm 13000 times faster than the software simulation. Besides, the hardware used a small maximum operating frequency compared to the software simulation.

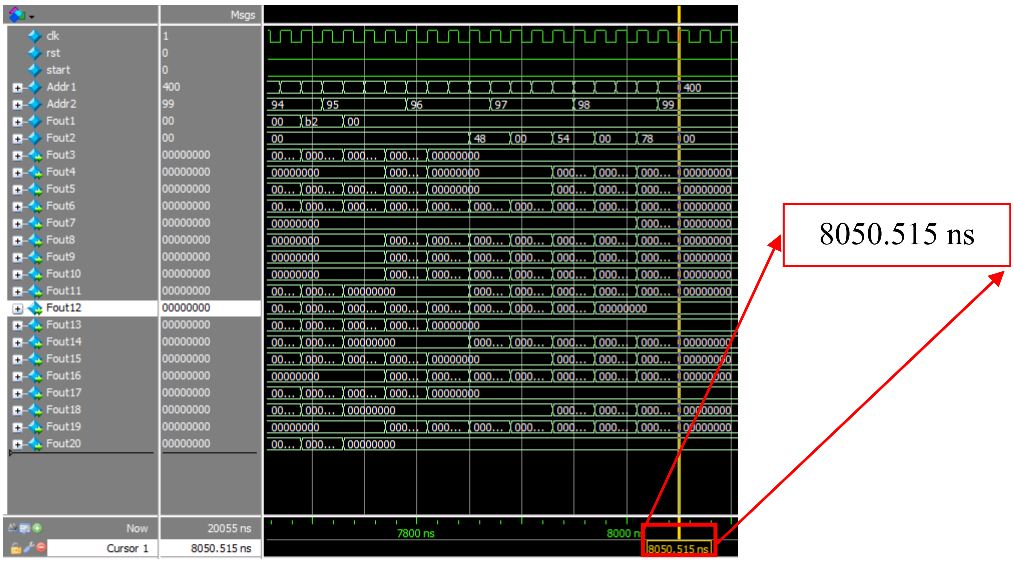

Fig. 9 *Simulation of the hardware implementation*

Table 2 *Performance comparison*

Performance metrics	Software	Hardware
Speed	104458 µs	8.05µS
Maximum operating frequency	2.60 GHz	157.41 MHz
Total logic elements	-	4,457
Total registers	-	2,522
Total memory bits	-	409,600
Total thermal power dissipation	-	71.73mW

4. Conclusion

This work proposes an interesting approach to address the limitations of software implementation for Convolutional Neural Networks (CNNs), particularly the slow processing speed and high-power consumption with large datasets and complex networks. By designing a dedicated hardware architecture for the ReLU activation function and mean pooling layer in CNN digit recognition using an FPGA, the work aims to achieve faster processing speeds and potentially lower power consumption compared to software implementations. Bypassing the bottleneck of a single ALU in a general-purpose processor, the custom hardware promises parallel processing capabilities, potentially accelerating computations. Reducing the overall energy footprint compared to software implementations could be crucial for battery-powered devices.

Acknowledgement

The author would like to thank the Faculty of Electrical and Electronic Engineering, University Tun Hussein Onn Malaysia for all the support throughout this work.

Conflict of Interest

The author declares that there are no conflicts of interest associated with this paper's publication. There are no any personal, business or professional ties that might influence this work. The research's neutrality and integrity are guaranteed by this transparency. The author promises to immediately disclose any future possible conflicts of interest that might surface.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Chessda Uttraphan, Araveindran Sithamparam;* ***data collection:*** *Chessda Uttraphan, Araveindran Sithamparam;* ***analysis and interpretation of results:*** *Chessda Uttraphan, Araveindran Sithamparam****; draft manuscript preparation:*** *Chessda Uttraphan, Araveindran Sithamparam. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Bhosle, Kavita & Musande, Vijaya. (2023). Evaluation of Deep Learning CNN Model for Recognition of Devanagari Digit. Artificial Intelligence and Applications. 1. 10.47852/bonviewAIA3202441.

[2] Ogundijo, Michael. (2023). Handwritten Digit Recognition Using Machine Learning Models : A Review. 10.13140/RG.2.2.12565.29925.

[3] Amin, Ruhul & Shin, Jungpil. (2023). A Fine-Tuned Hybrid Stacked CNN to Improve Bengali Handwritten Digit Recognition. Electronics. 12. 10.3390/electronics12153337.

[4] Ali, Saqib & Li, Jianqiang & Pei, Yan & Aslam, Muhammad & Shaukat, Zeeshan & Azeem, Muhammad. (2020). An Effective and Improved CNN-ELM Classifier for Handwritten Digits Recognition and Classification. Symmetry. 12. 15. 10.3390/sym12101742.

[5] Savita, Ahlawat & Choudhary, Amit & Nayyar, Anand & Singh, Saurabh & Yoon, Byungun. (2020). Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN). Sensors. 20. 3344. 10.3390/s20123344.

[6] Shyam, Radhey & Khanna, Shilpi & Verma, Priyanka & Maurya, Sakshi. (2023). Assessing the Performance of DL Methods in Handwritten Digit Recognition. 1. 2023. 10.37591/IJDSS.

[7] H. Prasetyo and B. A. P. Akardihas, “Batik image retrieval using convolutional neural network,” TELKOMNIKA (Telecommunication, Computing, Electronics and Control), vol. 17, no. 6, pp. 3010–3018, 2019, doi: 10.12928/telkomnika.v17i6.12701

[8] Ruiz, Camilo & Romero-Garcés, Adrián & Gonzalez Garcia, Martin & Marfil, Rebeca & Bandera, Antonio. (2023). FPGA-Based CNN for Eye Detection in an Iris Recognition at a Distance System. Electronics. 12. 4713. 10.3390/electronics12224713.

[9] Shao, Haijian & Ma, Edwin & Zhu, Ming & Deng, Xing & Zhai, Shengjie. (2023). MNIST Handwritten Digit Classification Based on Convolutional Neural Network with Hyperparameter Optimization. Intelligent Automation & Soft Computing. 36. 3595. 10.32604/iasc.2023.036323.

[10] Yan, Fei & Zhang, Zhuangzhuang & Liu, Yinping & Liu, Jia. (2022). Design of Convolutional Neural Network Processor Based on FPGA Resource Multiplexing Architecture. Sensors. 22. 5967. 10.3390/s22165967.