

## Article

# A Configurable and Fully Synthesizable RTL-Based Convolutional Neural Network for Biosensor Applications

Pervesh Kumar<sup>1</sup>, Huo Yingge<sup>1</sup>, Imran Ali<sup>1,2</sup> , Young-Gun Pu<sup>1,2</sup>, Keum-Cheol Hwang<sup>1</sup> , Youngoo Yang<sup>1</sup>, Yeon-Jae Jung<sup>1,2</sup>, Hyung-Ki Huh<sup>1,2</sup>, Seok-Kee Kim<sup>1,2</sup>, Joon-Mo Yoo<sup>1,2</sup> and Kang-Yoon Lee<sup>1,2,\*</sup>

<sup>1</sup> Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16416, Korea; itspervesh@skku.edu (P.K.); yingge@skku.edu (H.Y.); imran.ali@skku.edu (I.A.); hara1015@skku.edu (Y.-G.P.); khwang@skku.edu (K.-C.H.); yang09@skku.edu (Y.Y.); yjjung@skaichips.co.kr (Y.-J.J.); gray.huh@skku.edu (H.-K.H.); seokkeekim@skku.edu (S.-K.K.); fiance2@g.skku.edu (J.-M.Y.)

<sup>2</sup> SKAIChips, Sungkyunkwan University, Suwon 16419, Korea

\* Correspondence: klee@skku.edu; Tel.: +82-31-299-4954

**Abstract:** This paper presents a register-transistor level (RTL) based convolutional neural network (CNN) for biosensor applications. Biosensor-based diseases detection by DNA identification using biosensors is currently needed. We proposed a synthesizable RTL-based CNN architecture for this purpose. The adopted technique of parallel computation of multiplication and accumulation (MAC) approach optimizes the hardware overhead by significantly reducing the arithmetic calculation and achieves instant results. While multiplier bank sharing throughout the convolutional operation with fully connected operation significantly reduces the implementation area. The CNN model is trained in MATLAB<sup>®</sup> on MNIST<sup>®</sup> handwritten dataset. For validation, the image pixel array from MNIST<sup>®</sup> handwritten dataset is applied on proposed RTL-based CNN architecture for biosensor applications in ModelSim<sup>®</sup>. The consistency is checked with multiple test samples and 92% accuracy is achieved. The proposed idea is implemented in 28 nm CMOS technology. It occupies 9.986 mm<sup>2</sup> of the total area. The power requirement is 2.93 W from 1.8 V supply. The total time taken is 8.6538 ms.

**Keywords:** convolutional neural network; biosensor; diseases classification; RTL-based design



**Citation:** Kumar, P.; Yingge, H.; Ali, I.; Pu, Y.-G.; Hwang, K.-C.; Yang, Y.; Jung, Y.-J.; Huh, H.-K.; Kim, S.-K.; Yoo, J.-M.; et al. A Configurable and Fully Synthesizable RTL-Based Convolutional Neural Network for Biosensor Applications. *Sensors* **2022**, *22*, 2459. <https://doi.org/10.3390/s22072459>

Received: 14 December 2021

Accepted: 21 March 2022

Published: 23 March 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

A biosensor is a device that is sensitive to biological substances and converts its concentration into an electrical signal for further processing and analysis [1]. Existing artificial intelligence (AI) biosensors have many limitations: (1) They require a large number of well-labeled data; (2) have poor flexibility, and (3) features extraction strongly depends on logic and accumulation. Due to these restrictions, the potency of traditional biosensors is limited by aspects of performance such as accuracy, timing, etc. [2,3]. During the past few years, with the promising research in deep learning, especially in CNN, these limitations can be overcome. Apart from traditional biosensor systems, CNN can meaningfully progress on extracting the different features. Therefore, based on CNN, a biosensor system can exploit the unlabeled data and the features are learned automatically by the network architecture. Hence CNN is a promising approach for biosensor applications and has also been vastly studied by existing works [4,5].

While achieving real-time performance, CNN-based techniques demand much more computation and memory resources than conventional methods. Therefore, an energy efficient CNN implementation is inevitable. Application specific integrated circuit (ASIC) and field-programmable gate arrays (FPGA) [6–9] based accelerators are promising alternates. ASIC-based study in [10,11] is proposed for the purpose of cost efficiency, energy, and throughput. Similarly, FPGA-based research [12–14] achieve better performance because of the parallel computation. Moreover, CNN chip implementation is categorized into two classes: (1) standard or traditional chips, and (2) neuro-chips. Standard chips are further

divided into two subclasses: (I) multi-processor, and (II) sequential accelerator. In multi-processor, multiple cores are integrated for CNN operation. The purpose is to perform parallel operations for decreasing the throughput, improving the system performance and data loading by two folds. With the sequential accelerator breakthrough, the CNN algorithm on-chip integration is implementable.

The neuro-chips are built with multi-electrode arrays; a kind of integrated circuit chip that performs better because of good connectivity and fast and parallel computation. It needs low power and occupies less memory compared to traditional chips [15–18]. The research associated with neuro-chips is further classified into three main methods: fully analog, fully digital, and mixed analog/digital methods. For the mixed analog/digital [19,20], there are two operational modes; one is neuron mode, and another is synapse mode. For the neuron mode, usually, the asynchronous digital is adopted to perform the spiking. Regarding the synapse mode, most studies use analog methods to carry out compact operations such as those discussed in [21–23]. But there are studies that use analog/digital hybrid circuits to achieve timing synapses [24]. The digital logic foundation is adopted for on-chip structure configuration and providing corresponding algorithms. Different CNN architectures are implemented as per requirements. When using only the analog approach, mem-resistors are the key technology for implementing any functionality, keeping and tuning the resistance state according to the supply changes. Therefore, through it and resistive switching memories, it can meaningfully give the chance to achieve neural networks performance and functionalities in a fully analog manner. In [25], different functionalities of the neural network are implemented and achieved good performance in terms of small area and less power consumption. For the fully digital methods, the hardware algorithms are designed to address the functionalities and performance of the neural networks. These studies adopt software for modeling and training the neural network, get trained weights and bias values, and use these parameters for hardware implementation [26,27].

Previously, all the proposed CNN hardware implementation architectures for classification purposes used architectural parallelism and parameter reuse approaches. As a result, less memory was required or all memory on-chip was accommodated, but the drawback was moderate accuracy. In [28] and some other related studies, they also adopted parallel computation for reducing the processing time, but the trade-off was a large implementation area, nominal accuracy, and limitation of the number of layers.

The motivation of this work is to design a reconfigurable RTL-based CNN architecture for a biosensor for disease detection applications. The proposed design is fully synthesizable and technology independent. The parallel computation of MAC operation is used to reduce the arithmetic and extensive calculations. The multiplier bank is shared among all the convolutional layers and fully connected layers to reduce the implementation area.

The rest of the paper is organized as follows: Section 2 introduces the proposed top architecture of CNN for biosensor applications. The overall proposed system structure. The structures and functions of the sub-blocks are introduced in Section 3. Section 4 lists the experimental results, which includes software modeling results and hardware simulations results. Layout, mathematical results, and comparison with other works are also summarized. Finally, the paper is concluded in Section 5.

## 2. Top Architecture

The top structure of the proposed RTL-based CNN hardware implementation is shown in Figure 1. The proposed idea is comprised of three parts: (1) CNN architecture modeling and training in MATLAB<sup>®</sup>; (2) External on-board memory; and (3) hardware implementation system based on RTL compiler. In MATLAB<sup>®</sup>, two operations are performed: (1) CNN architecture is modeled, trained, tested, and trained weights and bias data is saved in a .txt file.; (2) Converting the input feature map data into binary data, which can be recognized by hardware tools and save it into a .txt file, for further processing.

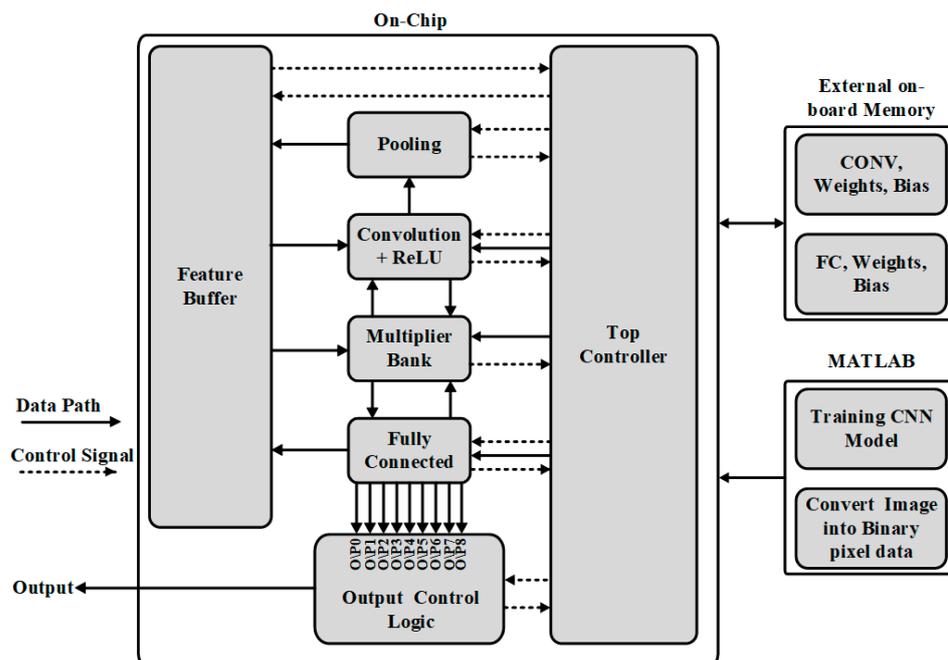
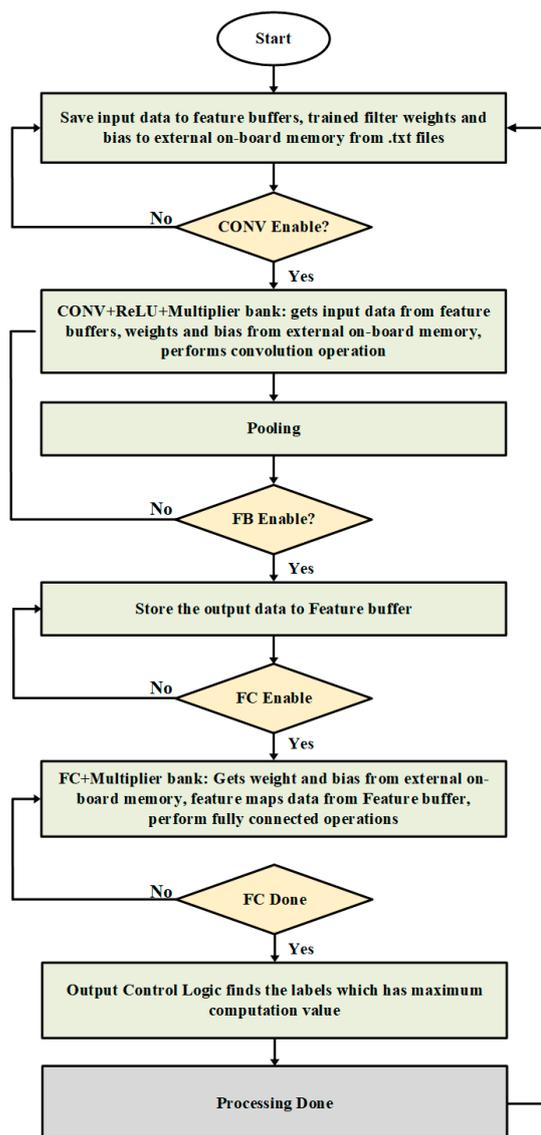


Figure 1. Top structure of CNN hardware implementation.

External on-board memory is a kind of multiple programmable memory. It is used for storing the trained kernels' weights and bias values with preloaded instruction for on-chip processing. Its operation is controlled by the top controller with reading instructions and enabling signals. The interface protocol is adopted to flow the data between external on-board memory and on-chip CNN system.

In the on-chip system, the same CNN architecture is modeled. Figure 1 shows the corresponding architecture, which is comprised of several building blocks. The dotted lines represent the data path and describe the data flow direction between different sub-blocks. The solid line shows the control signal working path and indicates how sub-blocks work together. On-chip block has several sub-blocks such as the memory system, which is designed to preload and store kernel weights, bias, and feature maps data. The CNN architecture layers are convolution + ReLU layer, pooling layers and fully connected (FC) layer. The top controller controls the whole system operation, connecting the different sub-blocks and controlling the data saving on the on-chip memory or going to the next stage operation. A multiplier bank is there to perform convolution calculations. Actually, it is the main computing resource for reducing the computation and is designed to be shared among all convolution + ReLU and fully connected layers. An output control logic is designed to find the final results of the system and also for final classification results.

The on-chip system operation is described in Figure 2. As the system starts, the feature buffer saves the input data, and the external on-board memory saves trained weights and bias. Once the CONV enable signal works, the CONV + ReLU and multiplier bank get input data from the feature buffers and weight and bias values from external on-board memory, and then the convolutional operation is performed. The convolved results go to pooling layer for sampling purposes. Once all convolution and pooling iterations are done, FB enables the signal works and the output activation feature maps are saved on the on-chip feature buffers. When FC enables the signal works, the FC layer and multiplier bank obtain weights and bias from external on-board memory and feature maps from feature buffers to perform the FC operation. As the FC operation finishes, the FC done signal is generated, the output control logic finds the labels having maximum computation value, and outputs it as a class.



**Figure 2.** Flow chart of the proposed CNN architecture.

Figure 3 shows the proposed CNN architecture. It has seven layers in total, out of which there are two convolutional layers, two pooling layers, two fully connected layers, and one softMax output layer. Input feature map data dimension is  $32 \times 32$ . In C1, the input feature maps are convolved with six kernels each of size  $5 \times 5$  with a stride of 1. It generates the six feature maps with a size of  $28 \times 28$ . Then it is handled by the activation function, such as rectified linear unit (ReLU). In S2, the size of feature maps has been sub-sampled to the half by adopting the max-pooling approach with size of  $2 \times 2$ , and the stride size is 2, to sub-sample the input feature maps to  $14 \times 14$ . In C3, 16 feature maps with  $14 \times 14$  is convolved with a 16 kernel of size  $5 \times 5$  to get 16 feature maps with the size of  $10 \times 10$  and again processed with ReLU operation. The S4 max-pooling operation is operated with size  $2 \times 2$ , the stride is 2 to achieve 16 feature maps with  $5 \times 5$ . The C5 is also a convolution layer with 120 kernels each of  $5 \times 5$  size, F6 is fully connected layer with 84 feature maps. Softmax is basically used for classification. The feature with the highest probability value is classified as an output result from handwritten digits. The CNN model is typically trained with a 32-bit floating point precision using MATLAB<sup>®</sup> platform. Since the MATLAB<sup>®</sup> computes parallelly, so the processing time is reduced compared to conventional C or C++ language processing approaches.

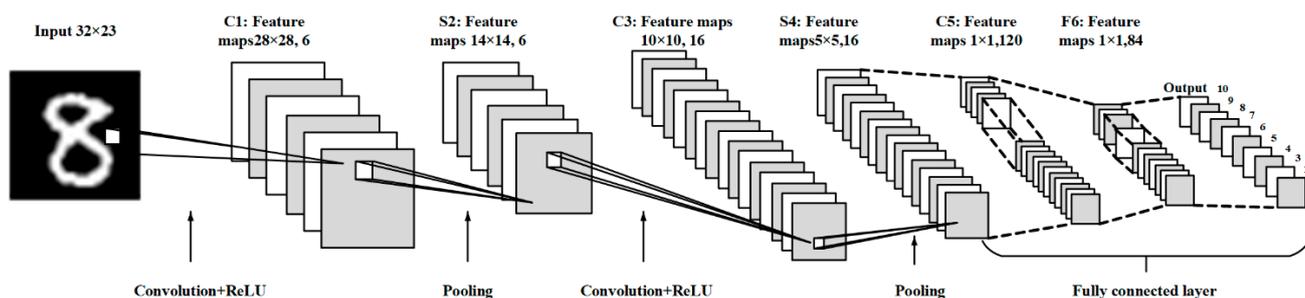


Figure 3. The proposed CNN model structure.

### 3. Building Blocks

#### 3.1. Top Controller

The top controller is the overall controlling module of the system. Firstly, it is used to control the module sequentially with enable and done signals. When these blocks need data, the enable signal for the next operation directly starts the next module, but also disables the current module operation. Secondly, the top controller is applied for communication with external memory for reading weights and bias information. Through the interface protocol, like serial peripheral interface (SPI), the data is transferred from external on-board memory to an on-chip system. Thirdly, the top controller is also used to differentiate the read/write indicates between the contiguous blocks and manage the calculation results being saved to the memories or go to the next stage. Fourthly, it is used to control convolution + ReLU module and share the multiplier bank with the fully connected module. It controls the selection of the data information in memories, which consists of various calculation blocks to be shared multipliers or pooling blocks, and also manages the multiplier calculated values that are being sent to the convolution operation or FC operation.

#### 3.2. Feature Buffers

The feature buffers, as shown in Figure 1, are used to save the output data of each sub-block. They are integrated to perform as on-chip memories. Each sub-block saves its output activation map into different on-chip memories, according to the size of the memories. These memories are built by the size of 10 k pf components, it means for each block, the capacity is 10 k bits. As for the case of the depth of the computation memory is bigger compared to this maximum size of 10 k, then another memory component is adopted which also has same size with this one.

#### 3.3. Convolutional Operation

The top architecture of the convolution layer is shown in Figure 4. It consists of a dedicated CONV controller, windows wrapper, multiplier bank, adder trees, and ReLU modules. The input feature maps data and kernel bias data for convolutional operation are pre-trained, after preloading, and saved in the on-chip memory. The calculation results from multiplier bank passed to adder trees for the next calculation. After computation, the results are transformed and stored into the on-chip feature buffers for next layer processing, which also has the same size with this one.

The CONV controller of convolution operation is mainly built by counters. Firstly it gets enable signal from the top controller and starts providing read signals for feature buffers and external on-board memory when convolution operation starts. Secondly, according to the kernel window size, it controls the window wrapper to select the input features maps data for partial convolution operation and slides this over all the spatial feature maps with the stride of 1. Finally, it manages the writing address to save output activation map value after ReLU processing to on-chip memories.

The role of the window wrapper is to select the window, as shown in Figure 5. According to the kernel size, selecting the corresponding pixels from the input feature map data. It consists of a window shifter and a window selector. The window shifter consists

of shift registers, as shown in Figure 6. After obtaining feature map data from the feature buffer and the shift signal from Conv Controller, it shifts the data serially and provides it to the window selector in parallel. The window selector consists of MUX, after getting kernel x and y coordinates from Conv Controller, it performs pixel selection, according to the kernel window size, as shown in Figure 7.

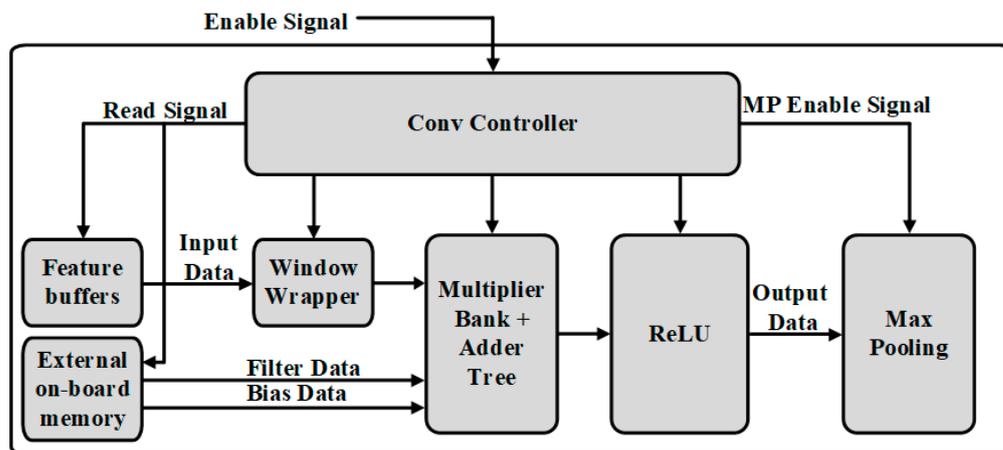


Figure 4. Overall architecture of the convolution layer.

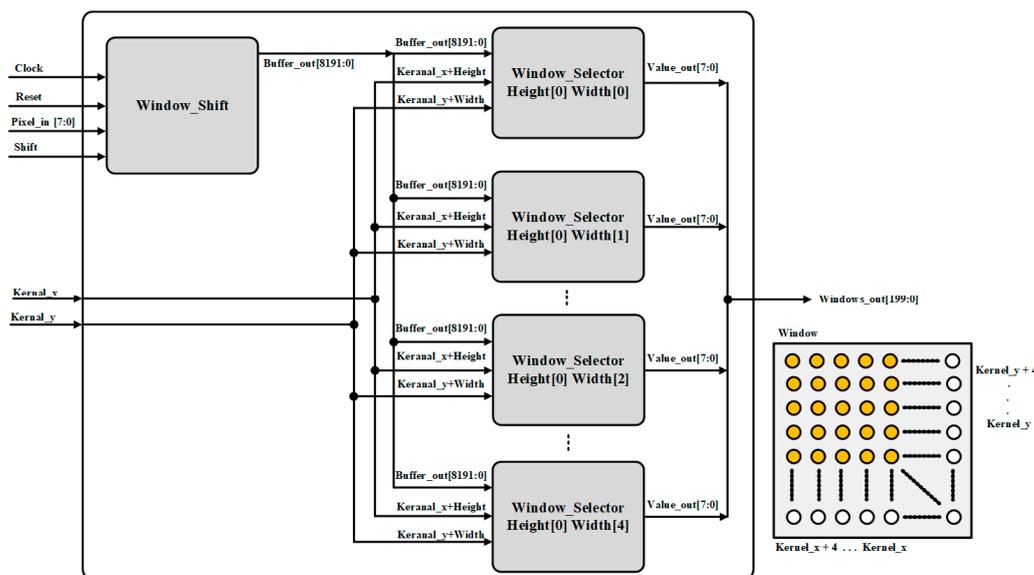


Figure 5. Window wrapper structure.

After receiving selected pixels from window wrapper and kernel, bias values from external on-board memory, the convolution operation is performed by multiplier bank and adder tree, as shown in Figure 8. The multiplier bank consists of multipliers, and the number of multipliers is decided by the kernel window size. For example, the kernel window size is  $5 \times 5 = 32$ . Each multiplier is used to multiply 8-bit kernel values with 8-bit selected pixel in a parallel manner and provide the result to the adder tree. The adder tree accumulates the multiplier bank result, also with bias values within one kernel window. The number of addresses is decided through the multipliers, then it can be calculated as follows in (1):

$$N_{adder} = \sum_{i=0}^{\log_2 N} 2^i + 1 \tag{1}$$

where,  $N$  is the number of multipliers, and  $N_{adder}$  is the number of addresses.

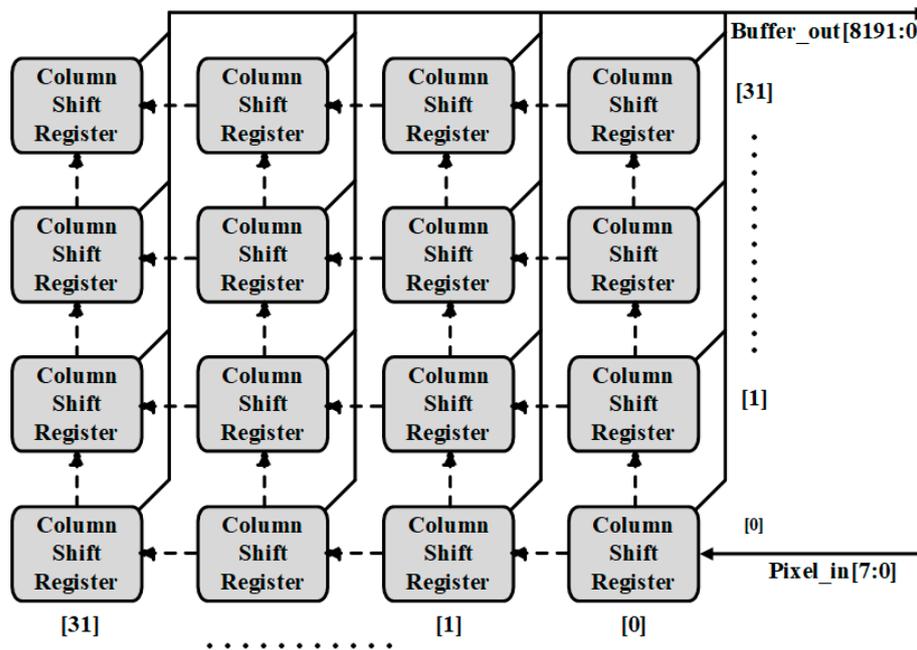


Figure 6. Window shift structure.

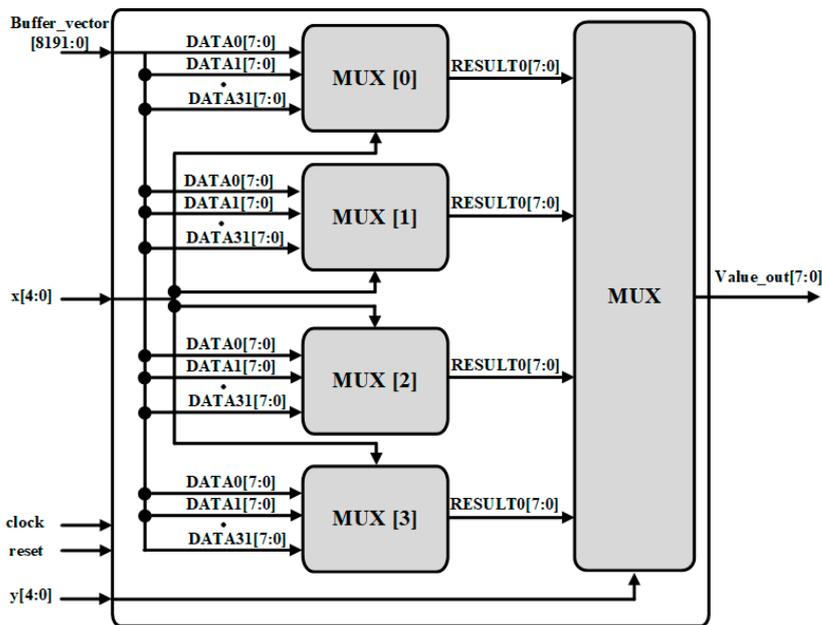


Figure 7. Window selector structure.

The block diagram of the ReLU operation is given Figure 9. The adder tree in this figure consists of comparator and mux, which performs as a kernel for the assigned bit of pixel value. Basically, it converts the negative values to zero, while it leaves the positive values unchanged. Mathematically this can be given as in (2):

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (2)$$

where  $f(x)$  represents the output of ReLU activation function. Its output is given to the max-pooling layer directly.

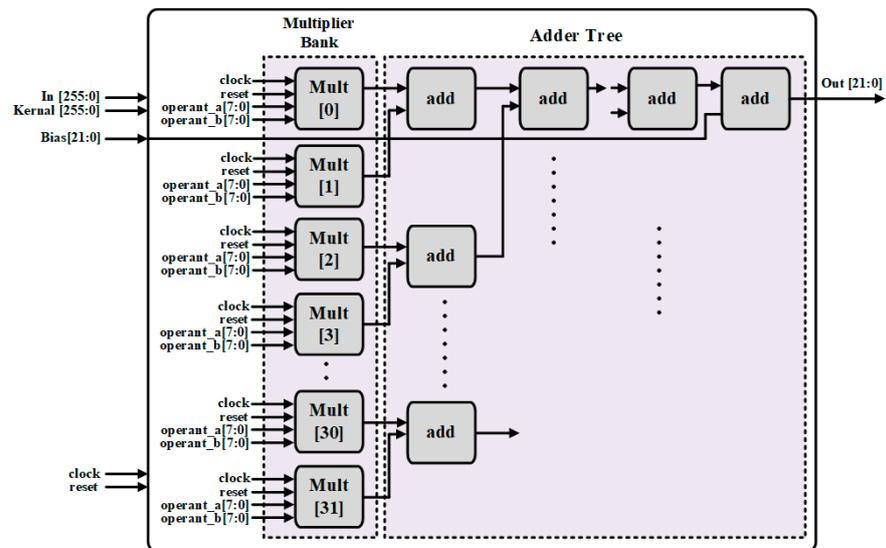


Figure 8. The architecture of convolutional operation.

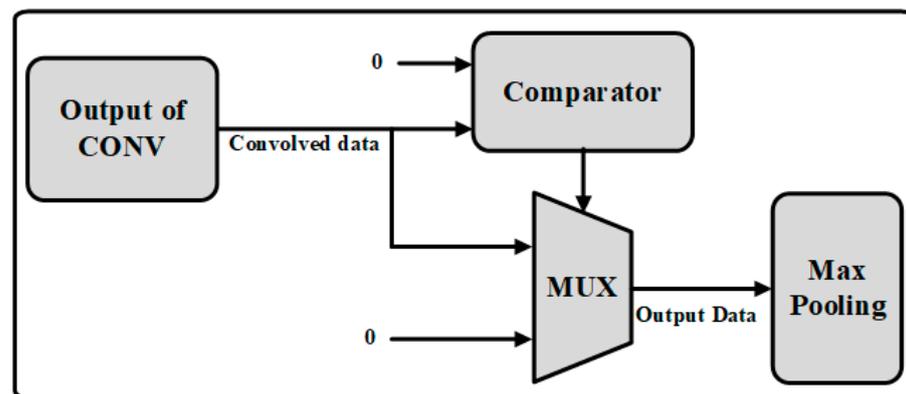


Figure 9. The architecture of convolutional operation.

### 3.4. Max-Pooling Operation

The Max-pooling operation is achieved by combining the max-pooling controller with comparators. The block diagram is given in Figure 10. After receiving the enable signal from the top controller and input values from the previous layer, the max-pooling controller performs partition of the input feature maps data into a set of rectangular sub-regions with the size of  $2 \times 2$ , and the stride size is 2. The difference with window selection of convolution operation is moved without any overlapping. The comparator is used to compare the  $2 \times 2$  sub-region values and outputs the maximum value of each sub-region. The controller also provides a read signal to the last stage for informing the start of operation and supplies a write address to save the output value to feature buffer for the next layer processing.

### 3.5. Fully Connected Operation

The function of the FC layer is the matrix multiplication. It is typically built by the the FC controller and multiplication and accumulation operations, which are similar to convolutional layers. A block diagram of fully connected operation is shown in Figure 11. After receiving enable signal from the top controller, the FC controller starts sending the read signal to the feature buffer and external on-board memory. It saves the last layer's computation results to obtain the input feature and weight value. The parallel multiplication and accumulation computations are used to calculate the sharing feature maps data to all the rows and they are calculated together in parallel. By adopting the

sharing multiplier bank with a convolutional layer, the computation is much reduced. Typically, the performance of the FC layer associates the input pitch points with output pitch points in the current layer. The function is given as in (3):

$$Y_i = \sum_{k=0}^{W*H-1} X_k \times W_i + b_i, N \geq i \geq 0 \tag{3}$$

where  $X_k$  represents the feature maps,  $Y_i$  represents calculations results,  $W_i$  is the weights values, and  $b_i$  represents the bias value.  $N$  is the number of the output nodes.

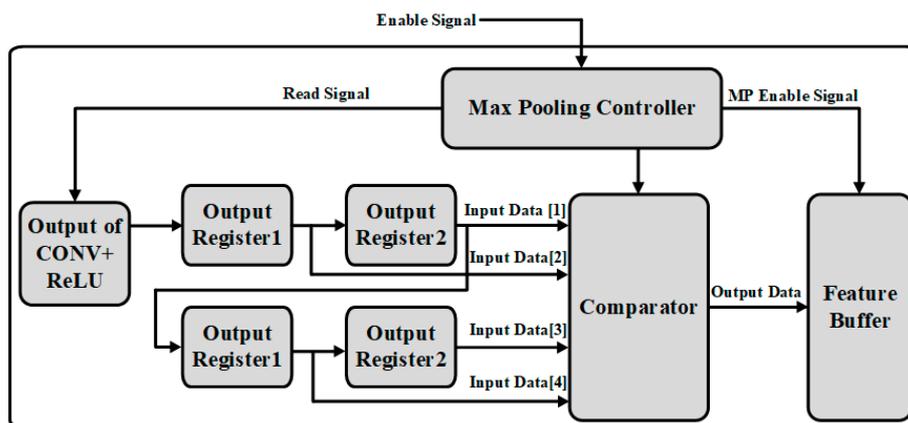


Figure 10. The architecture of the max-pooling operation.

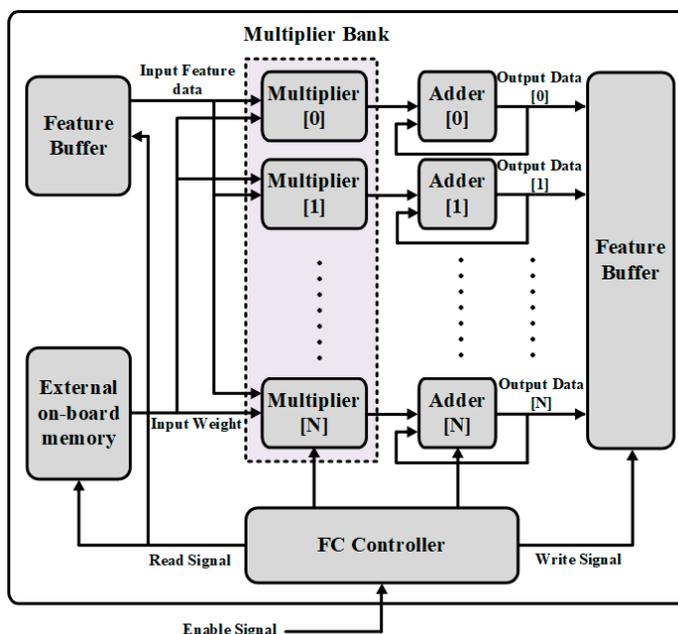


Figure 11. The architecture of the fully connected operation.

## 4. Experimental Results

### 4.1. MATLAB® Modeling and Results

The proposed CNN architecture was modeled and verified in MATLAB®. The model structure comprehensive analysis is given in Figure 12. It shows the layer-wise execution details, operations, operands and the total number of parameters at each layer after training. The model was trained on 60,000 images of MNIST® handwritten digits [29], number of epochs were 10, with batch size of 50. The learning rate was kept 0.5. The proposed CNN model consisted of 3 convolutional layers, and 2 fully connected layers with a kernel size of  $3 \times 3$  are used for each convolutional layer, ReLu activation function was used and

max-pooling with a strip size of  $2 \times 2$  was used. The model was tested on 1000 images from MNIST<sup>®</sup> data set. Initially, a digits image is given to the model. After processing, the training error is shown in Figure 13a and the accuracy is shown in Figure 13b. We achieved 92.4977% model training accuracy. The classification results are displayed in Figure 14.

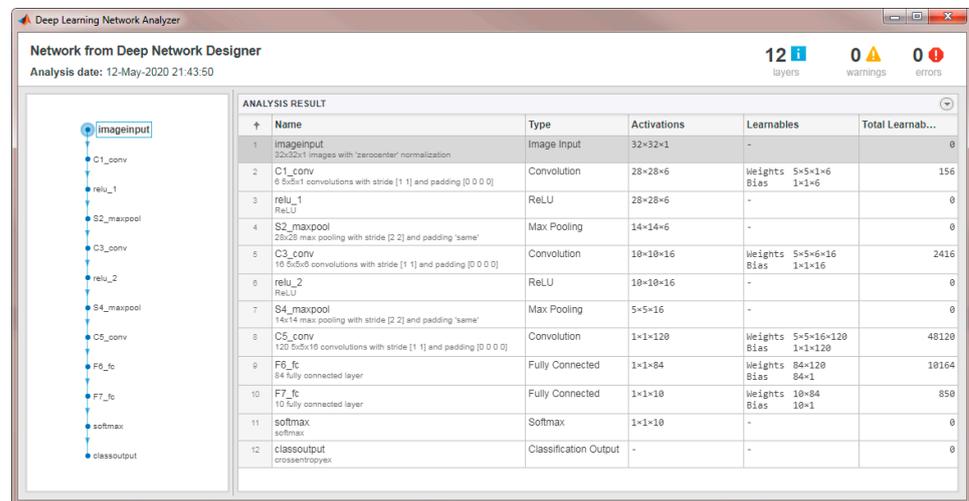


Figure 12. Analysis results of proposed CNN structure in MATLAB<sup>®</sup>.

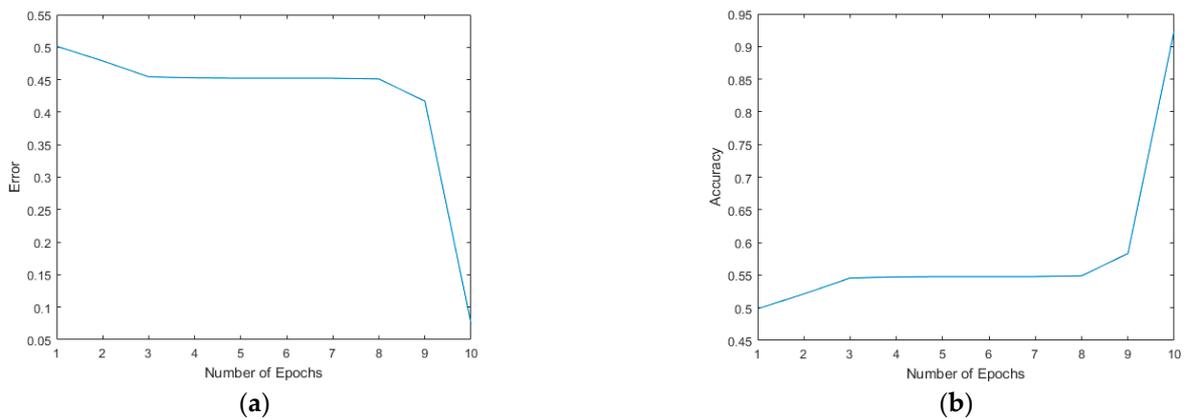


Figure 13. Training, testing error and accuracy of the proposed architecture. (a) The training error result. (b) The training accuracy.

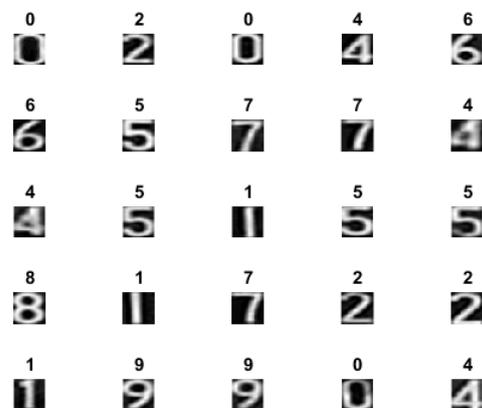


Figure 14. The classification result of the proposed CNN model.

#### 4.2. FPGA Implementation

The simulation result of the top convolutional layer is shown in Figure 15. Image data was preloaded into image array memory; after top window selection, the selected pixels could perform the convolution operation with kernel and bias value. Figure 16 basically shows the convolution operation logic, which combined the multiplier bank and adder tree to perform operation given in Equation (4).

$$Out = \sum in \times kernel + bias \quad (4)$$

where  $in$  is the input feature map data of convolution module,  $kernel$  is the corresponding weights data, and  $bias$  represents the system bias data.

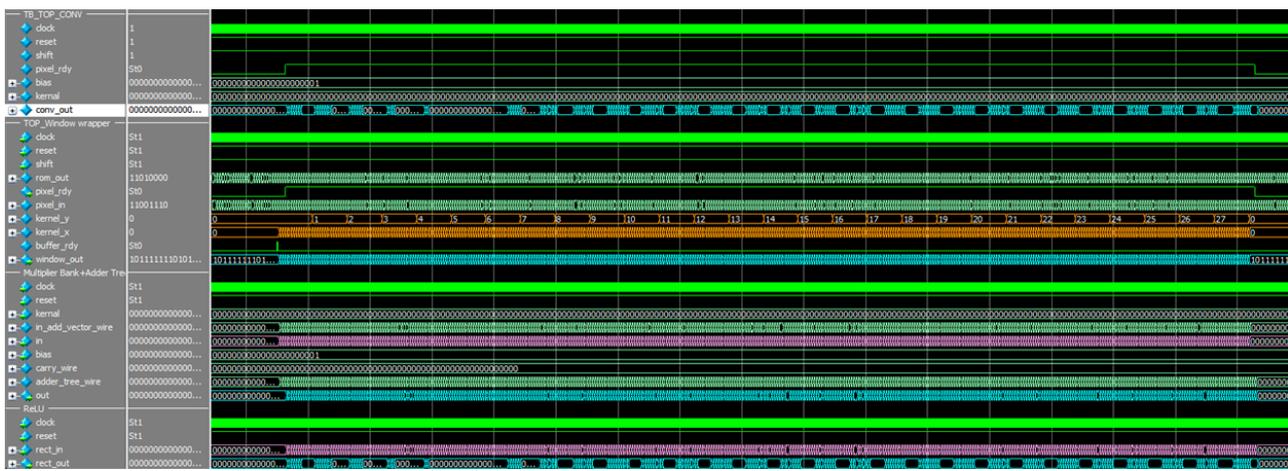


Figure 15. The simulation result of the top convolutional layer.

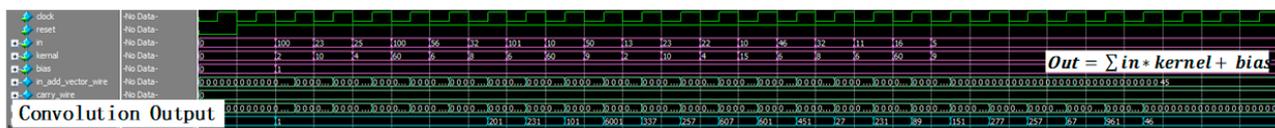


Figure 16. The simulation result of the convolution operation.

Window wrapper simulation results are described in Figure 17, which performed the pixels selection. As Figure 17a shows, according to the  $kernel_x$ ,  $kernel_y$ , the kernel window can slide around the whole image data and output the selected value which has the same size with kernel window, such as  $5 \times 5$ , Figure 17b shows the whole image data array with selected windows.

Max-pooling operations simulation results: Figure 18 shows the max-pooling operation results. When enable signal is asserted, the comparator compares the  $2 \times 2$  sub-region values and outputs the maximum value of each sub-region. Its calculation formula is given as in (5):

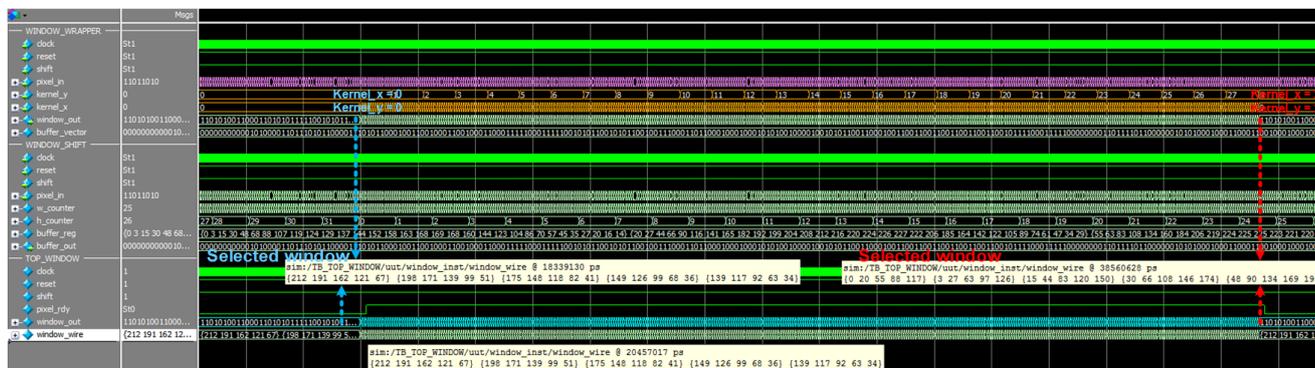
$$Out = MAX(input1, input2, input3, input4) \quad (5)$$

where  $Out$  represents the output value of the comparison.  $Input1$ ,  $input2$ ,  $input3$ ,  $input4$  are the four input values of each sub-region.

Fully connected operations simulation results: Figure 19 shows the FC module operation results. FC block associates input value with output value in the present module. When enable signal is high, it performs matrix multiplication. Shown as follow in (6).

$$sum_b = \sum feature_{pixels} * weight + bias \quad (6)$$

where,  $feature_{pixels}$  represents the input data of the fully connected module,  $weight$  represents kernel weights value,  $bias$  is the bias value, and  $sum_b$  presents the output value of the fully connected module.



(a)



(b)

Figure 17. The simulation result of the max-pooling operation, where windows wrapper results are shown in (a) Simulation result of window wrapper, while data of one selected image is shown in (b) One image data with selected window.

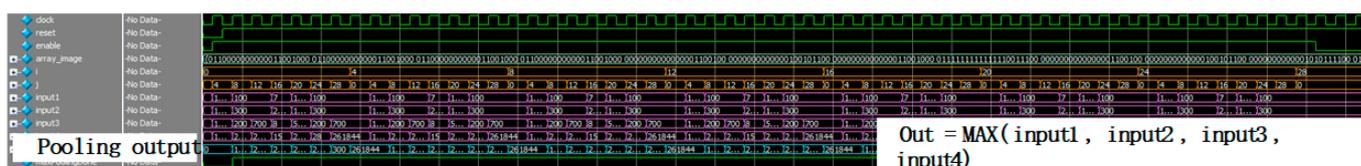


Figure 18. The simulation result of the max-pooling operation.

The top simulation results of the CNN system are described in Figure 20. Compared 10 outputs of final fully connected layer, the maximum value can be found which represents final classification results. It can be achieved by (7),

$$Y = \text{MAX}(a1, a2, a3, a4, a5, a6, a7, a8, a9) \tag{7}$$

where Y is the final output of the classification results. The fully connected operation results are a1–a9. Figure 20a–d shows the classification results when the input digit is 3, 5, 6, 8 separately.

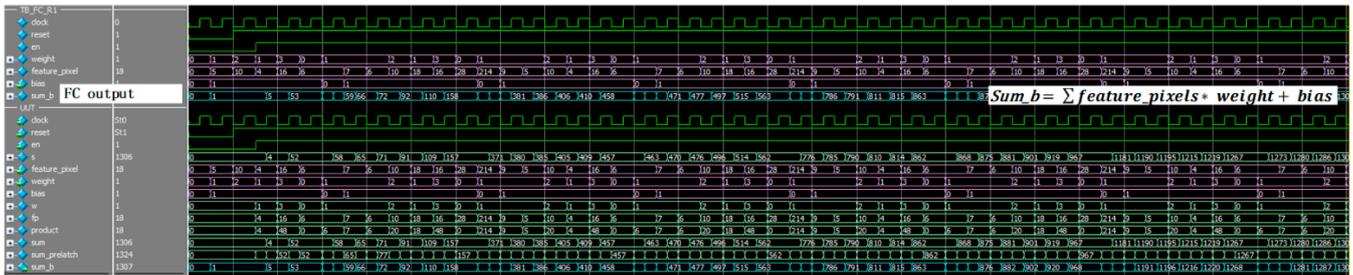
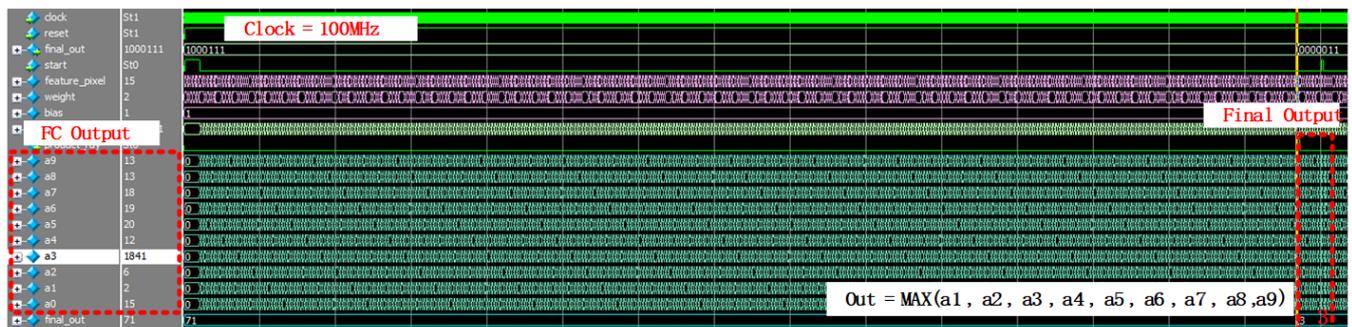


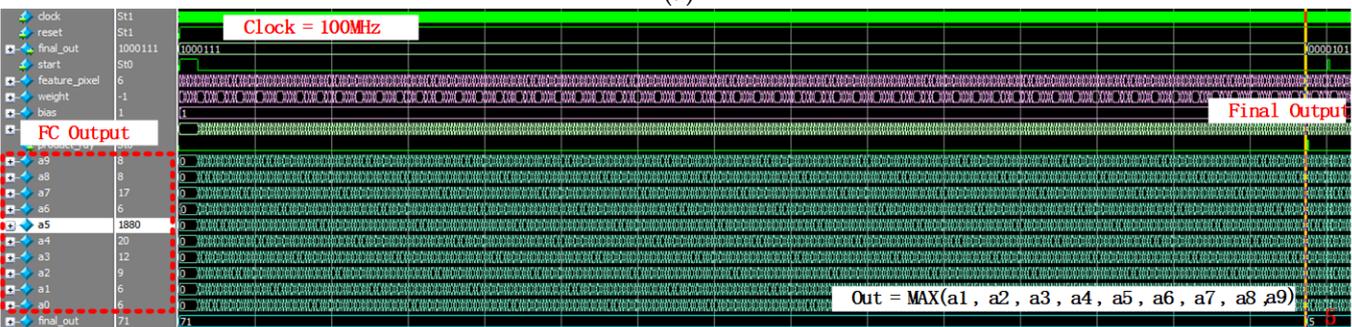
Figure 19. The simulation result of fully connected operation.

Figure 21 shows the timing consumption of different layers of the proposed CNN system. According to this table, CONV3 has the highest processing time because in this layer it has the largest number of kernels. So, the timing of loading weights and bias value to this layer and the feature map data loading time is the most costly. The total processing timing is 8.6538 ms.

Figure 22 shows the layout results of the proposed CNN on the chip logic part. It is implemented on 28 nm process technology by design compiler and IC compiler. The synthesis area is 3.16 mm × 3.16 mm.

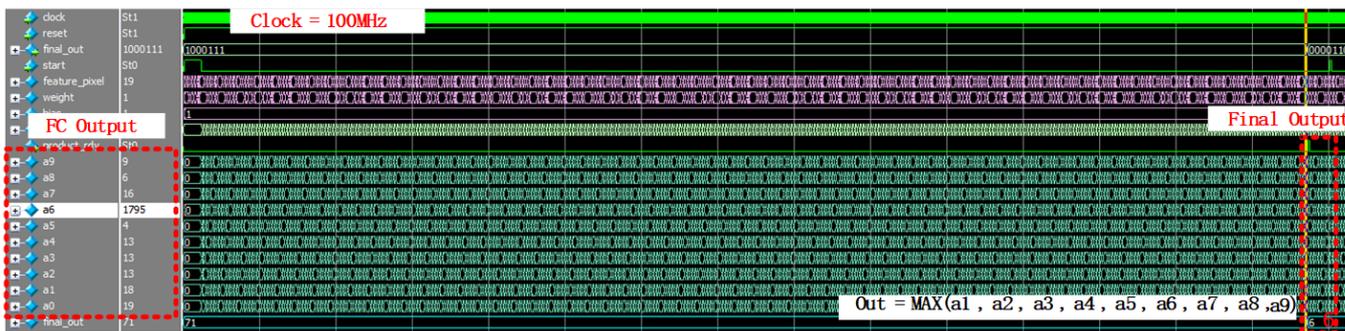


(a)

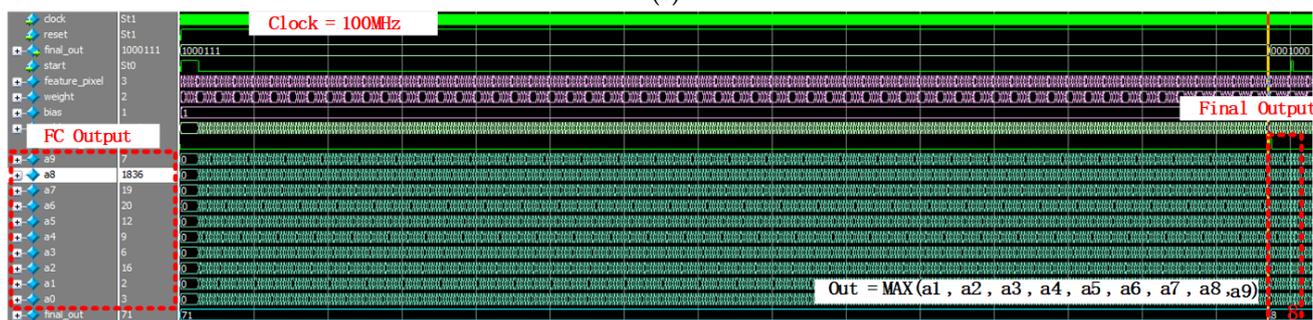


(b)

Figure 20. Cont.



(c)



(d)

Figure 20. The top simulation results of the proposed CNN system. (a) The top simulation result of digit 3. (b) The top simulation result of digit 5. (c) The top simulation result of digit 6. (d) The top simulation result of digit 8.

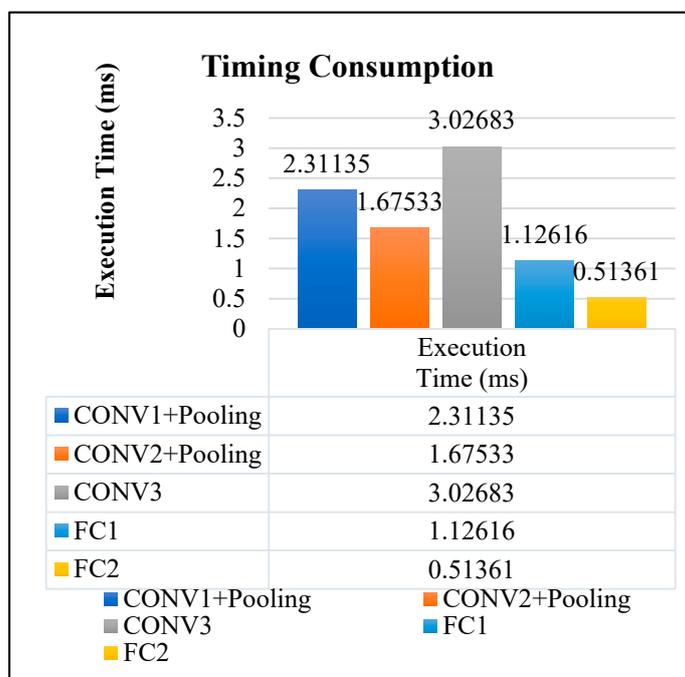


Figure 21. The timing consumption of different layers.

The proposed CNN system is verified on the FPGA. Figure 23 shows the experimental setup for measuring the proposed CNN system. Figure 23a shows the block diagram of the measurement framework, Figure 23b shows the actual verifying lab setup situation. UART Data Logger is the software for monitoring activities of ports. It monitors data exchanged

between the FPGA and an application via UART external interface, and analysis of the result for further researching. The FPGA board is connected to the computer by UART cable. After processing, the result is shown on the 7-segment on the FPGA board.

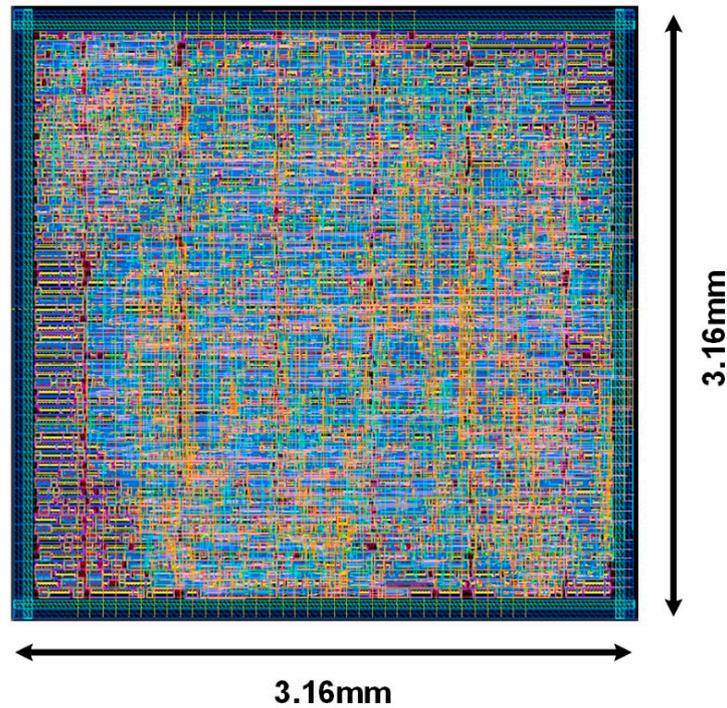


Figure 22. The layout of the proposed CNN system.

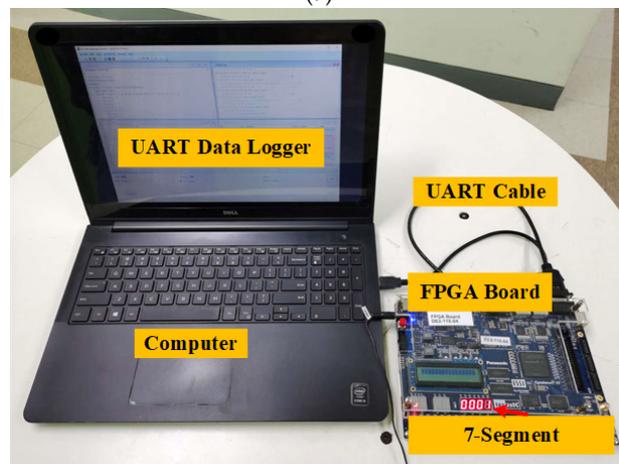
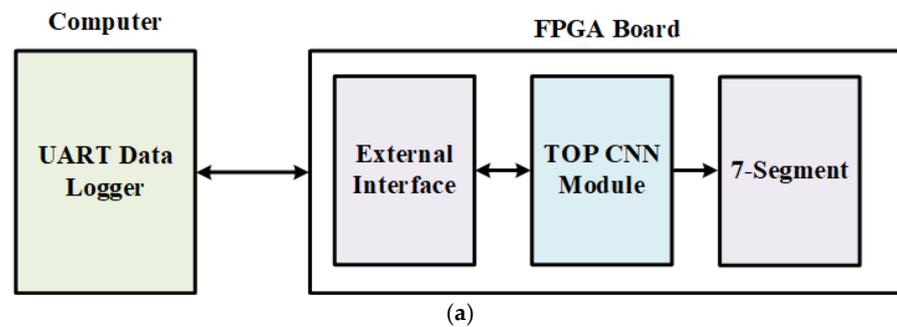


Figure 23. The proposed CNN system measurement setup. (a) The proposed CNN system measurement setup block diagram. (b) Measurement setup on FPGA.

Table 1 shows the performance summaries. Compared to the other three works, [9–11], firstly, we can achieve the highest classification accuracy. Secondly, the on chip memory size is relatively small due to adopting the methods of sharing the multiplier bank and adder tree, especially compared to [10], which has a smaller number of layers but has a large on chip memory size. Thirdly, the power consumption is relatively low compared to the other works, which are also a fully digital-based design.

**Table 1.** Performance comparison.

Parameter	This Work	[17]	[12]	[16]
Process (nm)	CMOS 28	CMOS 28	CMOS 65	CMOS 40
Architecture	Digital	Digital and Analog	Digital	Digital and Analog
Design Entry	RTL	-	RTL	-
Frequency (MHz)	100	300	550	204
CNN Model	6 layers	11 layers	9 layers (CNN/MLP)	-
Datasets	MNIST	CIFAR-10	MNIST	MNIST
V (V)	1.8	0.8	1	0.55–1.1
Power(W)	2.93	0.000899	0.00012	25
Accuracy (%)	92	86.05	98	98.2
On-Chip Memory	10 Kb	2676 Kb	-	-
Off-Chip Memory	40 Kb	no	-	-
Throughput (FPS)	5.33 k	-	8.6 M	1 k
Chip Area (mm <sup>2</sup> )	9.986	5.76	15	-

## 5. Conclusions

In the recent past, DNN and CNN have gained significant attention. This is because of its high precision and throughput. In the field of biosensors, there is still a gap in terms of the rapid detection of diseases. In this paper, we presented a synthesizable RTL-based CNN architecture for disease detection by DNA classification. The opted approach of MAC technique optimizes the hardware system by decreasing the arithmetic calculation and achieves a quick output. Multiplier bank sharing among all the convolutional layer and fully connected layer significantly reduce the implementation area.

We trained and validated the proposed RTL-based CNN model on MNIST handwritten dataset and achieved 92% accuracy. It is synthesized in 28 nm CMOS process technology and occupies 9.986 mm<sup>2</sup> of the synthesis area. The drawn power is 2.93 W from 1.8 V supply. The total computation time is 8.6538 ms. Compared to the reference studies, our proposed design achieved the highest classification accuracy while maintaining less synthesis area and power consumption.

**Author Contributions:** Conceptualization, P.K. and H.Y.; methodology, P.K. and I.A.; software, P.K. and H.Y.; validation, investigation P.K. and I.A.; resource H.Y.; data curation, P.K.; writing—original draft presentation, P.K.; writing-review and editing, P.K. and K.-Y.L.; visualization, P.K.; supervision, K.-Y.L., I.A., Y.-G.P., K.-C.H., Y.Y., Y.-J.J., H.-K.H., S.-K.K. and J.-M.Y.; project administration, K.-Y.L.; funding acquisition, K.-Y.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2021R1A4A1033424), and the Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2019-0-00421, Artificial Intelligence Graduate School Program (Sungkyunkwan University)).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Justino, C.I.L.; Duarte, A.C.; Rocha-Santos, T.A.P. Recent progress in biosensors for environmental monitoring: A review. *Sensors* **2017**, *17*, 2918. [CrossRef]
2. Schackart, K.E., III; Yoon, J.-Y. Machine Learning Enhances the Performance of Bioreceptor-Free Biosensors. *Sensors* **2021**, *21*, 5519. [CrossRef]
3. Cui, F.; Yue, Y.; Zhang, Y.; Zhang, Z.; Zhou, H.S. Advancing biosensors with machine learning. *ACS Sens.* **2020**, *5*, 3346–3364. [CrossRef]
4. Nguyen, N.; Tran, V.; Ngo, D.; Phan, D.; Lumbanraja, F.; Faisal, M.; Abapihi, B.; Kubo, M.; Satou, K. DNA Sequence Classification by Convolutional Neural Network. *J. Biomed. Sci. Eng.* **2016**, *9*, 280–286. [CrossRef]
5. Jin, X.; Liu, C.; Xu, T.; Su, L.; Zhang, X. Artificial intelligence biosensors: Challenges and prospects. *Biosens. Bioelectron.* **2020**, *165*, 112412. [CrossRef] [PubMed]
6. Shawahna, A.; Sait, S.M.; El-Maleh, A. FPGA-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access* **2019**, *7*, 7823–7859. [CrossRef]
7. Farrukh, F.U.D.; Xie, T.; Zhang, C.; Wang, Z. Optimization for efficient hardware implementation of CNN on FPGA. In Proceedings of the 2018 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA), Beijing, China, 21–23 November 2018; pp. 88–89.
8. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J. Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 1354–1367. [CrossRef]
9. Guo, K.; Zeng, S.; Yu, J.; Wang, Y.; Yang, H. A survey of FPGA-based neural network interface accelerator. *ACM Trans. Reconfig. Technol. Syst.* **2018**, *12*, 2.
10. Jiang, Z.; Yin, S.; Seo, J.S.; Seok, M. C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism. *IEEE J. Solid-State Circuits* **2020**, *55*, 1888–1897. [CrossRef]
11. Ding, C.; Ren, A.; Yuan, G.; Ma, X.; Li, J.; Liu, N.; Yuan, B.; Wang, Y. Structured Weight Matrices-Based Hardware Accelerators in Deep Neural Networks: FPGAs and ASICs. *arXiv* **2018**, arXiv:1804.11239.
12. Kim, H.; Choi, K. Low Power FPGA-SoC Design Techniques for CNN-based Object Detection Accelerator. In Proceedings of the 2019 IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, New York, NY, USA, 10–12 October 2019.
13. Mujawar, S.; Kiran, D.; Ramasangu, H. An Efficient CNN Architecture for Image Classification on FPGA Accelerator. In Proceedings of the 2018 Second International Conference on Advances in Electronics, Computers and Communications, Bangalore, India, 9–10 February 2018; pp. 1–4.
14. Ghaffari, A.; Savaria, Y. CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration. *Electronics* **2020**, *9*, 2200. [CrossRef]
15. Kang, M.; Lee, Y.; Park, M. Energy Efficiency of Machine Learning in Embedded Systems Using Neuromorphic Hardware. *Electronics* **2020**, *9*, 1069. [CrossRef]
16. Schuman, C.D.; Potok, T.E.; Patton, R.M.; Birdwell, J.D.; Dean, M.E.; Rose, G.S.; Plank, J.S. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *arXiv* **2017**, arXiv:1705.06963.
17. Liu, B.; Li, H.; Chen, Y.; Li, X.; Wu, Q.; Huang, T. Vortex: Variation-aware training for memristor X-bar. In Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, 8–12 June 2015; pp. 1–6.
18. Chang, J.; Sha, J. An Efficient Implementation of 2D convolution in CNN. *IEICE Electron. Express* **2017**, *14*, 20161134. [CrossRef]
19. Marukame, T.; Nomura, K.; Matusmoto, M.; Takaya, S.; Nishi, Y. Proposal analysis and demonstration of Analog/Digital-mixed Neural Networks based on memristive device arrays. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5.
20. Bankman, D.; Yang, L.; Moons, B.; Verhelst, M.; Murmann, B. An Always-On 3.8  $\mu\text{J}$ /86% CIFAR-10 Mixed-Signal Binary CNN Processor with All Memory on Chip in 28 nm CMOS. *IEEE J. Solid-State Circuits* **2018**, *54*, 158–172. [CrossRef]
21. Indiveri, G.; Corradi, F.; Qiao, N. Neuromorphic Architectures for Spiking Deep Neural Networks. In Proceedings of the 2015 IEEE International Electron Devices Meeting (IEDM), Washington, DC, USA, 7–9 December 2015; pp. 421–424.
22. Chen, P.; Gao, L.; Yu, S. Design of Resistive Synaptic Array for Implementing On-Chip Sparse Learning. *IEEE Trans. Multi-Scale Comput. Syst.* **2016**, *2*, 257–264. [CrossRef]
23. Hardware Acceleration of Deep Neural Networks: GPU, FPGA, ASIC, TPU, VPU, IPU, DPU, NPU, RPU, NNP and Other Letters. Available online: <https://itnesweb.com/article/hardware-acceleration-of-deep-neural-networks-gpu-fpga-asic-tpu-vpu-ipu-dpu-npu-rpu-ntp-and-other-letters> (accessed on 12 March 2020).

24. Pedram, M.; Abdollahi, A. Low-power RT-level synthesis techniques: A tutorial. *IEE Proc. Comput. Digit. Tech.* **2005**, *152*, 333–343. [[CrossRef](#)]
25. Ahn, M.; Hwang, S.J.; Kim, W.; Jung, S.; Lee, Y.; Chung, M.; Lim, W.; Kim, Y. AIX: A high performance and energy efficient inference accelerator on FPGA for a DNN-based commercial speech recognition. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 1495–1500.
26. Krestinskaya, O.; James, A.P. Binary Weighted Memristive Analog Deep Neural Network for Near-Sensor Edge Processing. In Proceedings of the 2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO), Cork, Ireland, 23–26 July 2018; pp. 1–4.
27. Hasan, R.; Taha, T.M. Enabling Back Propagation Training of Memristor Crossbar Neuromorphic Processors. In Proceedings of the 2014 International Joint Conference on Neural Networks (IJCNN), Beijing, China, 6–11 July 2014; pp. 21–28.
28. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; pp. 161–170.
29. Yann, L. The MNIST Database of Handwritten Digits. 1998. Available online: <http://yann.lecun.com/exdb/mnist/> (accessed on 1 January 2022).