JOIV : Int. J. Inform. Visualization, 6(1) - March 2022 53-59

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Design of a Low Area Digit Recognition Accelerator Using MNIST Database

Joonyub Kwon [a], Sunhee Kim [a,*]

[a] *Department of Semiconductor System Engineering, Sangmyung University, 31, Sangmyeongdae-gil, Dongnam-gu, Cheonan-si, Chungcheongnam-do, 31066, Republic of Korea*

Corresponding author: [*]happyshkim@smu.ac.kr

***Abstract*—Deep neural networks, a field of artificial intelligence, have been used in various fields. Deep learning is processed on high-performance GPUs or TPUs. It requires a high cost as much as its good performance. As the demand for edge computing increases, many studies have been conducted to perform complex deep learning operations in a low-computing processor. Among them, a typical study is to lighten the deep learning network. This paper proposes a handwritten digit recognition hardware accelerator suitable for edge computing using the MNIST database. After setting the correct rate for MNIST to 94% and performing network lighting processes, a hardware structure that can reduce the area of hardware and minimize memory access is proposed. The network is set as a two-layer, fully connected network. The network is modeled with Python and lightened while checking the performance. Network parameters, weights, and biases are quantized. The pixel number and bit number of MNIST input data are also reduced. The number of MAC units and the processing order of the hardware accelerator are determined so that there are no used MACs while performing the MAC operations in parallel. It is designed with Verilog HDL, and its functions are checked in ModelSim. And then, it is implemented in Xilinx Zynq ZC-702 to verify the operations. The designed number recognition accelerator is expected to be widely used in edge devices by reducing the area and memory access.**

***Keywords*— MNIST; accelerator; digit recognition; edge computing; fully-connected network.**

Manuscript received 22 Oct. 2021; revised 1 Nov. 2021; accepted 15 Dec. 2021. Date of publication 31 Mar. 2022.

I. INTRODUCTION

Deep neural networks, which is a field of artificial intelligence (AI), are being used in various fields such as speech recognition, image classification, disease diagnosis, autonomous driving, image generation, and complex games [1]–[4]. In some fields, the accuracy of AI has reached a level that surpasses that of humans. The accuracy of AI presupposes very high costs: mass storage, high-speed parallel computing, etc. Google DeepMind's AlphaGo, which won against the legendary Sedol Lee by 4-1 in March 2016, is known to use 48 Tensor Processing Units (TPUs). TPUs are hardware devices specialized to parallelize vector/matrix calculations, compared to graphics processing units (GPUs). Therefore, now, methods for network optimization, such as computational convolution optimization, activation function improvement, parameter factorization, network pruning, and network quantization, are being studied as well as performance improvement [5]–[9].

Recently, as the demand for edge computing increases, research on dedicated accelerators instead of high-performance processors such as GPUs and TPUs has been conducted [10]–[12]. In edge computing, edge AI devices process data in real-time, and the collected data are sent to a central server [13]–[15]. Improvement, including network training, is made on a central server. In other words, most edge AI devices only inference without training. In addition, most edge AI devices run on batteries. Therefore, the area and power consumption of the devices is as important as the performance [16]. Although 64/32-bit floating-point numbers are used based on GPU/TPU-software [17]–[19] to find network parameters through training, 16/8/4/2 fixed-point numbers are used in inference to reduce area and power consumption [20]–[22]. In addition, many efforts have been made to reduce area and power consumption, such as reducing memory storage capacity and memory access by compressing data [23]–[25].

The MNIST database (Modified National Institute of Standards and Technology database) is a database of

handwritten digits [26]. Each data is a digit image between 0 and 9, and the digit is located in the center of the 28x28 black and white image. It consists of 60,000 training sets and 10,000 test sets. It is a representative database widely used in neural networks. As a result, many studies using the MNIST database have been conducted, and the accuracy reaches about 99.8% [27]–[29]. Most of the networks with an accuracy of 99% or more use a convolutional neural network (CNN) [30], [31]. CNN consists of a convolutional layer, an activation layer, a pooling layer, and a fully connected layer. It is difficult for even people to accurately judge the digit figures in which these networks made errors [32]. Recently, rather than improving the performance of MNIST, many studies have been conducted to reduce complexity and increase usability so that it can be processed even in small devices [33], [34].

This paper investigates the handwritten digits detector using the MNIST database as a dedicated accelerator that can be used in edge AI devices. It focuses on area and power consumption rather than accuracy. The next section explains network model lightning methods and hardware structure. In section III, we show implementation and results and then conclude.

II. Materials and Method

A. Network model optimization

In this study, a neural network model optimization is performed with the goal of a neural network with two fully connected layers for the MNIST database. The network model is coded in Python to train and test. First, the two-dimensional input images are converted into one-dimensional arrays. Since they are image data, when they are changed to one-dimensional arrays, the spatial information of the images is lowered. However, it is possible to reduce the complexity of the convolution calculation and power consumption due to memory access.

The second lighting method is to reduce input data size to reduce hardware area and power consumption. The MNIST database consists of images of 728 (=28x28) pixels. In this study, the size of input data is reduced by half in the horizontal and vertical directions, respectively, and a 14x14 pixel image is created. In other words, the number of pixels of the input image data decreases by a quarter. To reduce the image size, pooling is required. Pooling methods include max pooling, min pooling, and average pooling. Max pooling selects the largest pixel value in a predetermined pixel area. Conversely, min pooling selects the smallest pixel in a predetermined pixel area. Averaging pooling selects the average value of the pixel values included in a predetermined pixel area.

To compare the three pooling methods, a 2-layer network is constructed, as shown in Fig. 1. The input layer consists of 196 input nodes and one bias node. The hidden layer has 14 nodes, and the output layer consists of 10 nodes. Since no operation is performed in the input layer, it is a 2-layer network consisting of one hidden layer and one output layer.

The input layer has 196 nodes (x0 ~ x195) because MNIST images are converted to one-dimensional arrays with 196 (=14x14) elements after max pooling with 2x2 filter and stride. The number of nodes in the hidden layer is set to 14. The number of nodes in the input layer, which is already set to 196, is the integer multiple of the number of nodes in the hidden layer. As will be explained later, if it is set to an integer multiple, it is to reduce the not-used multipliers when processing multiplication in parallel.

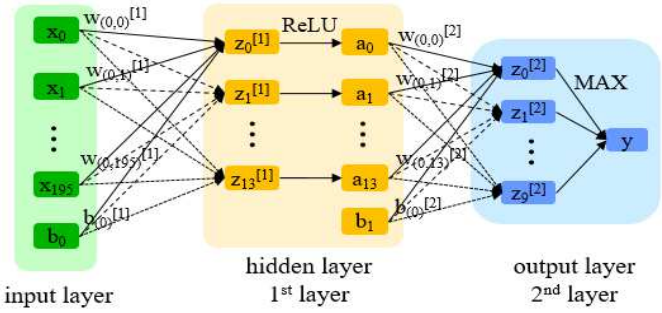

Fig. 1 Two-layer network model

In the hidden layer, each node applies a linear transformation to the input data x_n through the weights $w_{(t,s)}$ and sums the bias b_0. This can be expressed as a formula as follows.

$$z_t^{[1]} = \sum_{s=0}^{195} \left(w_{(t,s)}^{[1]} x_s \right) + b_0^{[1]} \quad (1)$$

where $w_{(t,s)}^{[n]}$ is the weight of the n-th layer and represents a weight connected from the s-node of the n-1 th layer to the t-node of the n-th layer. That is, $w_{(t,s)}^{[1]}$ is the weight of layer 1, and is the weight connected from the s node of the input layer to the t node of the first layer. $b_0^{[1]}$ means the bias is connected to the first layer, and there is only one node in one layer. And $z_t^{[n]}$ means the value after multiply-accumulate (MAC) is performed at the t-th node of the n-th layer.

The ReLU function is used as the activation function of the hidden layer. Among the activation functions, the sigmoid function is widely used in logistic classification because it outputs a value between 0 and 1. However, the sigmoid function has a disadvantage in that the optimization process is slow. In addition, the larger the absolute value of x, the greater the possibility of losing the differential value during gradient backpropagation. The hyperbolic tangent function, tanh, solves the problem of slowing down in the sigmoid optimization process by shifting the function's center point to zero. However, the vanishing gradient problem in which the derivative value disappears above a certain value for the differential function remains [35]. The ReLU function is a function to solve the gradient vanishing problem of sigmoid and tanh. If x is greater than 0, the slope is 1, and if it is less than 0, the value of the function becomes 0. It is characterized by faster learning, less computational cost, and simpler implementation than sigmoid and tanh functions. Leaky ReLU and PReLU have been developed to compensate for the disadvantage of dying ReLU in which neurons can die when their value is less than 0. However, they have similar performance to ReLU and have the disadvantage of being complex. Therefore, the ReLU function is selected as the activation function. The MAC calculation result of the hidden layer $z^{[1]}$ passes through the activation function ReLU, and the result value $a^{[1]}$ is transmitted to the output layer.

The output layer is composed of 10 nodes because it represents the probabilities for each of the 10 digits from 0 to 9. In the output layer, as in the hidden layer, each node applies

a linear transformation to the input data a_n through the weights $w_{(t,s)}^{[2]}$ and sums the bias $b_1^{[2]}$. In the output layer, the final result value y is obtained through the Max function, which selects the largest value of given input data instead of the ReLu function.

TABLE I
COMPARISON OF NETWORK ACCURACY ACCORDING TO BATCH SIZE

batch size	10	100	1000
train images	91.53%	93.59%	94.07%
test images	91.50%	93.26%	93.96%

The network model is coded using Python. After training 10,000 times while changing the mini-batch size of the MNIST data, the accuracy of the training data and the test data is compared. As shown in Table I, the performance difference after 100 was less than 1%. Therefore, the mini-batch size is set to 100 and then trained 10,000 times.

Since MNIST data is a black and white image, max pooling is expected to perform much better than the other two pooling methods, but there is a difference of about 0.1% in accuracy. As a result, the image size is reduced by max pooling.

TABLE II
COMPARISON OF NETWORK ACCURACY ACCORDING TO THE NUMBER OF BITS OF WEIGHTS AND BIASES

	float	9bits	8bits	7bits	6bits	5bits
accuracy	94.08%	94.08%	94.03%	94.01%	93.67%	92.87%

Third, the number of bits of weights and biases is optimized. In general, GPUs are used to train with datasets, so float 32 or float 16 data types are used. However, if the network is not very deep and the data input itself is 8 bits as in this study, good performance can be obtained even if weights and biases are defined as a fixed-point data type. Table II and Fig. 2 show the results of comparing accuracy while changing the number of bits of weights W and biases b from float 32 to fixed point 4 bits. For float 32, the accuracy is about 94%. When changing to a fixed point and reducing the number of bits from 9 bits to 4 bits, if it is more than 7 bits, it shows about 94% accuracy, which is similar to the case of float 32. And, when it is lowered to 6 bits, it can be seen that the accuracy drops sharply. We select 8 bits for weights and biases to reduce the number of bits while maintaining the accuracy to a certain degree.

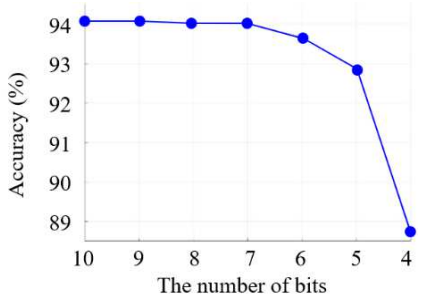

Fig. 2 Comparison of Network Accuracy according to the Number of Bits (4-10 bits) of Weights and Biases

Fourth, the number of bits of input data is optimized. The input data is the pixel data of MNIST. Since it is a black (255) and white(0) image, its number of bits is 8. As summarized in Table III, the accuracy is checked by lowering the number of input data bits from 8 bits to 3 bits. Accuracy differs by about 0.1% from 8 bits to 3 bits. When its number of bits is 4, the accuracy of the test image is the highest. Therefore, in this study, the number of bits for the handwritten-digit image is selected to be 4 to optimize the hardware area. As a result, the size of one image is reduced by 1/8 from 6,272 (=28x28x8) bits to 784 (=14x14x4) bits. This can reduce the required memory storage and reduce memory access, which can reduce power consumption as well as area.

TABLE III
COMPARISON OF NETWORK ACCURACY ACCORDING TO THE NUMBER OF BITS OF AN INPUT DATA PIXEL

	8bits	7bits	6bits	5bits	4bits	3bits
accuracy	93.98%	93.97%	93.98%	93.98%	94.03%	93.97%

B. *Hardware architecture*

An accelerator consists of a controller, a MAC block, a ReLU block, and a Max block. The controller sequentially executes the entire hardware process. The MAC block is responsible for the operation of the affine layer. The ReLU block acts as an activation function of the first layer, and the final number recognition is made in the MAX block.

In order to implement the proposed two-layer network in hardware, the following four structures are compared in terms of low area and low power consumption. For explanation, we will use the T_m symbol, which means the time it takes from the start of one multiplication to the start of the next multiplication in the m-th structure. After the first multiplication, it becomes T. After the second multiplication operation, it becomes 2T, and after the n-th multiplication operation, it becomes nT.

As shown in Fig. 3, the first structure has multipliers as many as the nodes of the input layer to take advantage of the parallel hardware processing. For hidden layer processing, 196 input data x are given as the first input of each multiplier. At the start of the operation, a weight $w_{(0,n)}$ is given to each multiplier M_n. That is, all multiplication operations for the first node of the hidden layer are performed at time T. To add all 196 multiplication results, adders with an 8(=$\log_2$(196))-step tree structure are used. When the accumulated result passes through the ReLU block, the result of the first node of the hidden layer is completed. The result is stored in the local register and waits until all calculations of the hidden layer are finished.

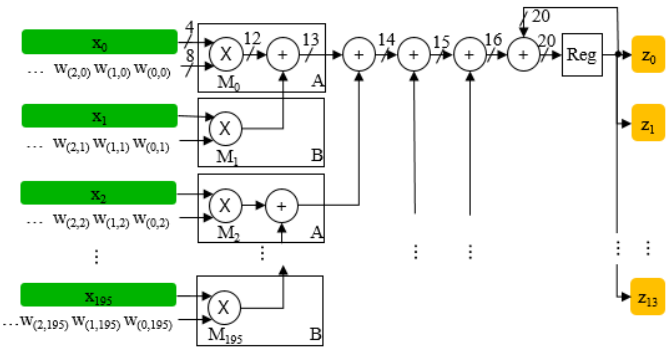

Fig. 3 The first structure of the accelerator MAC

Next, while x is fixed, $w_{(1,n)}$ is given to each multiplier M_n, and all multiplication operations for the second node of the hidden layer are performed. Similarly, the result of the second

node of the hidden layer is completed through the 8-step tree structure adders and the ReLU block. When this process is repeated 14 times, the operation on the hidden layer is finished. During this time, the 196 input data are accessed from memory at once, and weights are accessed from memory 14 times by 196. To store the 196 input data and results, 196 4-bit local registers and 14 25-bit local registers are required.

The output layer is calculated using $w^{[2]}$, b_1 and the results of the hidden layer. The number of the hidden layer results is 14, and the output layer has 10 nodes. That is, a total of 140(=14x10) multiplication operations are required. Since there are 196 MACs for the hidden layer, the output layer can complete the calculation at one T. In summary, this structure has 196 multipliers, and all operations are completed in 15T. All input data and weight are accessed only once from a memory. Although the processing speed is fast, it is not suitable for a low-area and low-power accelerator because there are too many multipliers.

In the second structure, there are 14 multipliers instead of 196 multipliers. In the hidden layer, as shown in Fig. 4, the one input data x_n is broadcast to 14 multipliers, and the 14 weights $w_{(t,n)}^{[1]}$ corresponding to the input are transmitted to 14 multipliers, respectively. As can be seen from the index of weights, 14 multiplication results are values for different nodes. 14 multiplication results are added in the first structure because they are all for one hidden layer node. However, in the second structure, since 14 multiplication results are results for different nodes, each result must be stored in the local register and accumulated until the rest of the multiplication is finished. Since one input data is processed at a time, the hidden layer operation is finished when 196T is reached. The 14 values stored in the local registers are input to the next output layer through the ReLU function.

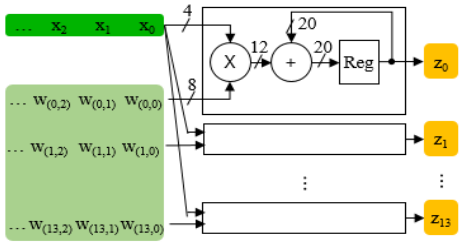

Fig. 4 The second structure of the accelerator MAC

Since the output layer has 10 nodes, 4 out of 14 MACs are not used when calculating the output layer. As in the hidden layer, x_n is input equally to 10 multipliers, and 10 weights $w_{(t,n)}^{[2]}$ corresponding to the input are connected to each multiplier. When this process is repeated 14 times, the output layer operation is finished. Totally, it takes 210(= 196+14)T. Memory access is performed only once for both input data and weights.

In order to eliminate not used multipliers when calculating the output layer, the third structure does not broadcast input data as shown in Fig.5. In the second structure, as input data are broadcast, each multiplier calculates the value transmitted to different nodes at a specific time. However, in the third structure, each multiplier uses different input data, so multipliers calculate values transmitted to the same node at a specific time. Therefore, since the result of one hidden layer node requires 196 multiplications, the result can be obtained at 14T. Since there are 14 nodes in the hidden layer, repeating this 14 times will get the results of all hidden layer nodes. That is in the second structure, the calculation of the hidden layer ends at 196T.

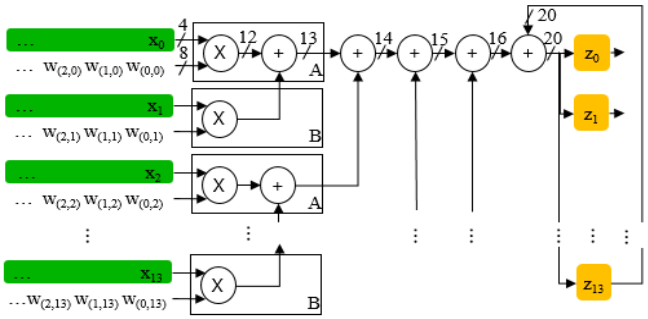

Fig. 5 The third structure of the accelerator MAC

When comparing the hardware structure, in the second structure, one adder is connected to each multiplier as shown in Fig. 4, but in the third structure, there are adders in a tree structure that adds all the results of 14 multiplication, as shown in Fig. 5. The second structure requires 14 20-bit adders, whereas, in the third architecture, there are 7 13-bit adders, 3 14-bit adders, 2 15-bit adders, one 16-bit adder, and finally, one 20-bit adder. That is, the number of adders is the same as 14, but since the number of bits of the adders of the third structure is smaller, the hardware area is also smaller.

In addition, in the output layer, since 14 different input data are connected to 14 multipliers to calculate the result of one node, not used multipliers do not exist. In addition, since the number of nodes in the output layer is 10, the calculation of the output layer can be completed by repeating the calculation only 10 times. That is, it takes 206(=196+10)T.

Compared to the second structure, the third structure can reduce the area and computation time. However, in the second structure, 196 input data must be read from memory only once, but in the third structure, 196 input data must be read from memory 14 times, that is, whenever the value of each node of the hidden layer is calculated. In other words, the third structure consumes more power than the second structure.

The fourth structure is proposed by supplementing the shortcomings of two structures, as shown in Fig. 6. The MAC of the fourth structure is the same as the MAC of the third structure, but the calculation order is different. During the first T, in both structures, 14 multipliers calculate the value for the first node of the hidden layer using the input data x_0~x_{13} and the weights $w_{(0,0)}^{[1]}$~$w_{(0,13)}^{[1]}$.

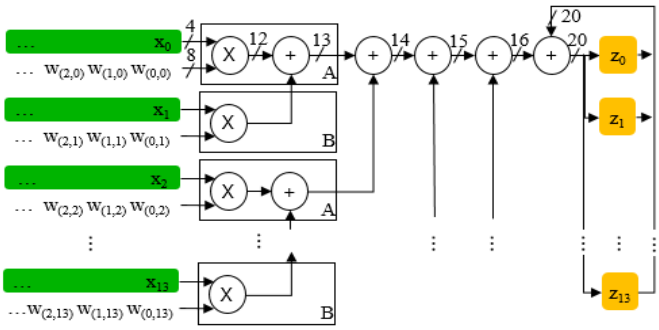

Fig. 6 The fourth structure of the accelerator MAC

In the third structure, the values for the first node of the hidden layer are calculated successively by changing both the

input data and the weights. This increases memory access. So, in the fourth structure, the first input data x_0~x_{13} are fixed, and the weights are changed from $w_{(0,0)}^{[1]}$~$w_{(0,13)}^{[1]}$ to $w_{(1,0)}^{[1]}$~$w_{(1,13)}^{[1]}$. Then, the sum of the multiplications calculated at the first T and the sum of the multiplications calculated at the second T become the values of different hidden layer nodes, so they are stored in different local registers. When this process is repeated 14 times, the multiplication of the input data x_0~x_{13} is finished. And then, the input data are changed to x_{14}~x_{27}. By repeating the input data from x_0~x_{13} to x_{182}~x_{195}, the results for all hidden layer nodes can be obtained. In summary, the fourth structure keeps the area small like the third structure and only accesses each data once like the second structure.

TABLE IV
COMPARISON OF AREA AND LATENCY OF THREE STRUCTURES

multiplier		adder	memory access	latency
2	14	20-bit 14 adders	3,082	210T
3	14	13-bit 7, 14-bit 3, 15-bit 2, 16-bit 1 and 20-bit 1 adders	5,684	206T
4	14	13-bit 7, 14-bit 3, 15-bit 2, 16-bit 1 and 20-bit 1 adders	3,082	206T

Table IV shows the comparison of multipliers, adders, and memory accesses for three structures. They have 14 multipliers in common. Two operands of the multiplier are 4-bit unsigned data and 8-bit signed data. Its result is 14-bit signed data. As described above, in terms of area, the third and fourth structures are better than the second structure, and in terms of memory accesses, the second and fourth structures are better than the third structure. In addition, the third and fourth structures are better than the second structure in terms of latency. Therefore, we make an accelerator for MNIST based on the fourth structure.

C. Hardware design

The proposed MNIST accelerator consists of a controller, a MAC block, a ReLU block, a MAX block, local registers, and memory. The ReLU block receives the MAC results from the hidden layer as inputs and determines output values with the ReLU function. The MAX block receives the MAC results from the output layer as inputs and selects the maximum value. While calculating the hidden layer and output layer, local registers store the node results of each layer until the calculation of each layer is completed. That is, in the hidden layer, the result of the first node comes out at the first T, and then it has to wait for 13T, so it is stored in a local register. When the calculation of the hidden layer is finished, the 14 values stored in local registers are again given as inputs to the MAC to calculate the output layer.

Weights and input data are stored in memory. The weight is 8 bits, and 196*14 bytes for the first layer and 14*10 bytes for the second layer are needed. However, since 14 bytes must be read at a time, the memory is physically composed of three 32-bit (4 bytes) and one 16-bit (2 bytes), and addresses are arranged as shown in Fig. 7. The bias is 14 +10 bytes and is stored between addresses 0xCE0 and 0xD4F. The input data (196*4 bits) is stored between addresses 0xD00 to 0xD6F.

The MAC block, which is the circuit with the fourth structure discussed above, consists of 14 multipliers and tree-structured adders. However, the number of bits of multipliers and adders is expanded. The number of bits of data used in the hidden layer and the output layer is different. In the previous description, the number of bits is decided according to the hidden layer. In the hidden layer, the inputs of MAC are 4-bit data and 8-bit weight and bias. However, since the inputs of the output layer are the ReLU results of the hidden layer, they are neither 4 bits nor 8 bits. The ReLU result is 20-bit data. Therefore, the two operands of the multiplier have 20 bits and 8 bits, respectively, and the subsequent adders are expanded by 16 bits each compared to the hidden layer adders. So, we simulate again and reduce the result of the hidden layer to 8 bits. Therefore, both multiplier operands are modified to 8 bits, and the adders are increased by 4 bits from the previous one. During multiplication and addition, the calculation is performed with extended bits, and only the final values, which are the input values of the ReLU and MAX blocks, are truncated to 8 bits. Since the input data of the hidden layer are fixed at 4 bits, 4-bit zeros are added to make 8-bit data. The structure of the modified MAC is shown in Fig. 8.

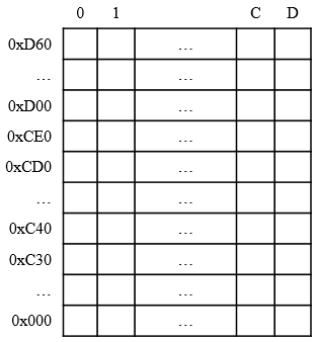

Fig. 7 Memory allocation

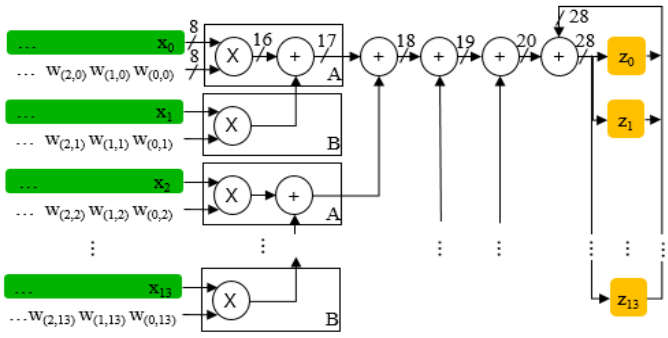

Fig. 8 Modified structure of the accelerator MAC

As seen in Equation (1), the bias values must be added to the MAC results before they are transferred to the ReLU or MAX block. However, as shown in Fig. 8, there is no adder for bias. This is because we use the bias value of each node as the initial value of local registers. Therefore, there is no need for additional hardware for bias, and latency can be reduced.

III. RESULTS AND DISCUSSION

The proposed accelerator was designed using Verilog-HDL and the functions were verified in ModelSim. First, state machines in the controller were confirmed. As shown in Fig. 9, three states L-state, X-state, and W-state were defined. L-state distinguishes the hidden layer and the output layer. X-

state indicates the state of the input data, and W-state indicates the state of the weight. When L-state has a value of 0, that is, during the hidden layer, X-state changes from 0 to 13(=b1101). And, while X-state maintains one state, W-state changes 14 times. In output layer, L-state is 2(=b10), and the X-state changes 10 times.

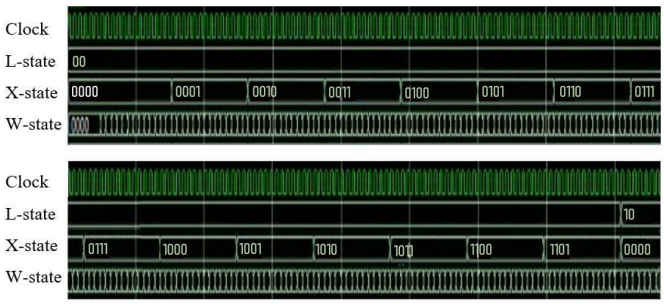

Fig. 9 Verilog simulation results for state machines

Fig. 10 shows input/output data as well as state machines. The 14x14 pixels modified MNIST handwritten data, 9 and 2, are sequentially input into the MAC. The input data are 196 (=14x14) one-dimensional arrays. However, 14 input data are read at a time because the number of multipliers is 14. As shown in Fig. 10, each row of the input image is read during the hidden layer processing. So, the simulation waveform appears as if the input image is rotated. Finally, it is confirmed that the input images are determined as '9' and '2', respectively.

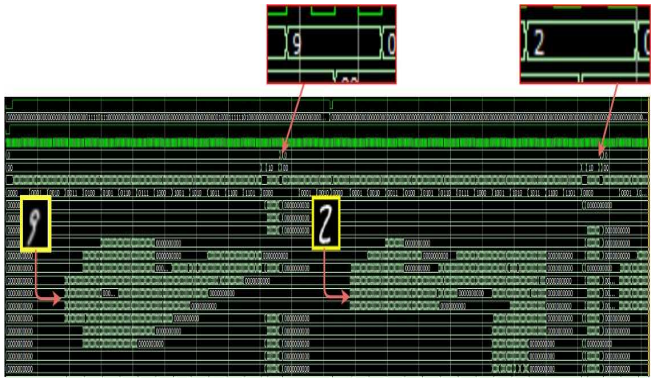

Fig. 10 Verilog simulation results for the accelerator

After function verification in ModelSim, it was implemented using Xilinx ZYNQ ZC-702. The ZC-702 board consists of a Processing System (PS) containing an ARM Cortex-A9 processor and an FPGA Aritix-7 Programmable Logic (PL) for user-designed logic. PS and PL communicate by AXI bus and support interrupt. So, AXI4-Lite slave interface was added to the designed accelerator and then implemented in PL of ZC-702 board.

The operation proceeds in the following order. The ARM processor writes weights, biases and input data in memory. It instructs the accelerator to start operation by setting the operation start register among the control registers of the accelerator. The accelerator starts operation by reading weight, bias, and input data from memory. When the operation is finished, the accelerator interrupts the processor to inform that the operation is completed. The processor reads a result value from the result value register. As shown in Fig. 11, the result value determined by the accelerator and the image written in the memory are displayed on terminal. In Fig. 11, the image is judged as "2", and it can be seen that the actually input image is also "2". That is, it is confirmed that it operates correctly.

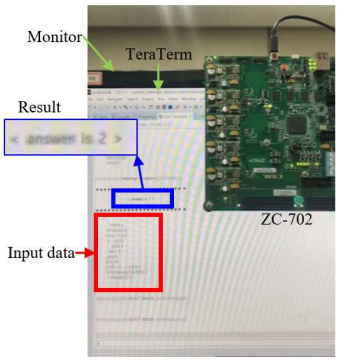

Fig. 11 Implementation and test

IV. Conclusion

In this paper, we designed a low-power, low-area, handwritten digit recognition deep learning accelerator. In general, iterative training is performed on high-performance GPUs to find the optimal deep learning network, and the optimized network model has high precision. However, in the case of a handwritten digit recognizer, it is possible to recognize digits accurately enough without using a high-performance GPU. Therefore, we conducted the training process in software and then went through the lighting process to become a low-power, low-area hardware accelerator while maintaining proper performance. The handwritten digit data MNIST is a 28x28 black and white image, and each pixel is 8 bits. The MNIST data was converted into a one-dimensional array to light the network, and the circuit complexity was lowered by reducing 784 pixels to 196 pixels. A 2-layer fully connected network was established, and the hidden layer consisted of 14 nodes. By lowering the number of bits of weights and biases, the recognition performance was analyzed, and weights and biases were set as 8-bit fixed points. Then, the number of input data bits was lowered, and the recognition performance was analyzed, and the value of input data was changed to a 4-bit integer. After confirming the network model, we designed the hardware and verified the operation by selecting a structure that can reduce memory access and area. The designed hardware showed 94% accuracy as predicted in the algorithm stage and maintains a level similar to that determined by humans. Therefore, it is expected to be widely used in edge devices that require low area and low power.

References

[1] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi and J. A. Benediktsson, "Deep Learning for Hyperspectral Image Classification: An Overview," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 9, pp. 6690-6709, Sept. 2019, doi: 10.1109/TGRS.2019.2907932.

[2] S.L. Oh, Y. Hagiwara, U. Raghavendra, R. Yuvaraj, N. Arunkumar, M. Murugappan and U. R. Acharya, "A deep learning approach for Parkinson's disease diagnosis from EEG signals," *Neural Comput & Applic.*. vol. 32, pp. 10927–10933, 2020. 10.1007/s00521-018-3689-5.

[3] L. Jiao and J. Zhao, “A Survey on the New Generation of Deep Learning in Image Processing,” *IEEE Access*, vol. 7, pp. 172231-172263, 2019. 10.1109/ACCESS.2019.2956508.

[4] N. Justesen, P. Bontrager, J. Togelius and S. Risi, “Deep Learning for Video Game Playing,” *IEEE Transactions on Games*, vol. 12, no. 1, pp. 1-20, March 2020. 10.1109/TG.2019.2896986.

[5] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and Quantization for Deep Neural Network Acceleration: A Survey,” *Neurocomputing*, vol. 461, pp. 370-403, Oct. 2021, 10.1016/j.neucom.2021.07.045.

[6] V. Lebedev, V. Lempitsky, “Speeding-up convolutional neural networks: A survey,” *Bulletin of the Polish Academy of Sciences. Technical Science*, vol. 66, no. 6, pp. 799-811, 2018, 10.24425/bpas.2018.125927.

[7] M. D. Zeiler, and R. Fergus. “Stochastic pooling for regualization of deep convolutional neural networks,” arXiv preprint arXiv:1301.3557, 2013, 10.48550/arXiv.1301.3557.

[8] J. Y. Wu, C. Yu, S. W. Fu, C. T. Liu, S. Y. Chien and Y. Tsao, “Increasing Compactness of Deep Learning Based Speech Enhancement Models With Parameter Pruning and Quantization Techniques,” *IEEE Signal Processing Letters*, vol. 26, no. 12, pp. 1887-1891, Dec. 2019. 10.1109/LSP.2019.2951950.

[9] J. Guo, W. Liu, W. Wang, J. Han, R. Li, Y. Lu, S. Hu, “Accelerating Distributed Deep Learning By Adaptive Gradient Quantization,” *in Proc. ICASSP*, Barcelona, Spain, 2020, pp. 1603-1607, doi: 10.1109/ICASSP40776.2020.9054164.

[10] C. Wang, L. Gong, X. Li, and X. Zhou, “A Ubiquitous Machine Learning Accelerator With Automatic Parallelization on FPGA,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2346-2359, 1 Oct. 2020. 10.1109/TPDS.2020.2990924.

[11] Y. Toyama, K. Yoshioka, K. Ban, S. Maya, A. Sai and K. Onizuka, “An 8 Bit 12.4 TOPS/W Phase-Domain MAC Circuit for Energy-Constrained Deep Learning Accelerators,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 10, pp. 2730-2742, Oct. 2019. 10.1109/JSSC.2019.2926649.

[12] Y. Wang, Y. Wang, H. Li and X. Li, “An Efficient Deep Learning Accelerator Architecture for Compressed Video Analysis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* pp. 1-1, 2021. 10.1109/TCAD.2021.3120076.

[13] J. Chen and X. Ran, “Deep Learning With Edge Computing: A Review,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655-1674, Aug. 2019. 10.1109/JPROC.2019.2921977.

[14] K. Cao, Y. Liu, G. Meng and Q. Sun, “An Overview on Edge Computing Research,” *IEEE Access*, vol. 8, pp. 85714-85728, 2020. 10.1109/ACCESS.2020.2991734.

[15] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, “Edge computing: A survey,” Future Generation Computer Systems, vol. 97, pp. 219-235, 2019. 10.1016/j.future.2019.02.050.

[16] S. W. Yang, “Efficient Deep Learning on Limited System Resources in FPGAs Performance Comparison on Floating Points,” M.S. thesis, Dept. Computer & Information Technology, Korea Univ., Seoul, Korea, 2019.

[17] A. Iwata, Y. Yoshida, S. Matsuda, Y. Sato, and N. Suzumura, “An artificial neural network accelerator using general purpose 24 bits floating point digital signal processors,” in IJCNN, Washington, DC, USA, vol. 2, 1989, pp.171–175, doi: 10.1109/IJCNN.1989.118695.

[18] J. Civit-Masot, F. Luna-Perejón, S. Vicente-Díaz, J. M. Rodríguez Corral and A. Civit, “TPU Cloud-Based Generalized U-Net for Eye Fundus Image Segmentation,” *IEEE Access*, vol. 7, pp. 142379-142387, 2019. 10.1109/ACCESS.2019.2944692.

[19] R. Murillo, A. A. D. Barrio, and G. Botella, “Deep PeNSieve: A deep learning framework based on the posit number system,” Digital Signal Processing, vol. 102, 102762, 2020. 10.1016/j.dsp.2020.102762.

[20] Q. H. Vo, N. Linh Le, F. Asim, L. W. Kim and C. S. Hong, “A Deep Learning Accelerator Based on a Streaming Architecture for Binary Neural Networks,” *IEEE Access*, vol. 10, pp. 21141-21159, 2022. 10.1109/ACCESS.2022.3151916.

[21] Y. Toyama, K. Yoshioka, K. Ban, S. Maya, A. Sai and K. Onizuka, “An 8 Bit 12.4 TOPS/W Phase-Domain MAC Circuit for Energy-Constrained Deep Learning Accelerators,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 10, pp. 2730-2742, Oct. 2019. 10.1109/JSSC.2019.2926649.

[22] H. F. Langroudi, Z. Carmichael, and D. Kudithipudi, “Deep Learning Training on the Edge with Low-Precision Posits,” arXiv preprint *arXiv.1907.13216*, 2019. 10.48550/arXiv.1907.13216.

[23] H. W. Son, D. Y. Lee, and H. W. Kim, “Compact CNN Accelerator Chip Design with Optimized MAC And Pooling Layers,” *Journal of the Korea Institute of Information and Communication Engineering*, vol. 25, no. 9, pp. 1158-1165, Sept. 2021, 10.6109/JKIICE.2021.25.9.1158.

[24] S. F. Hsiao, K. C. Chen, C. C. Lin, H. J. Chang, and B. C. Tsai, “Design of a Sparsity-Aware Reconfigurable Deep Learning Accelerator Supporting Various Types of Operations,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 376-387, Sept. 2020. 10.1109/JETCAS.2020.3015238.

[25] L. Kang, H. Li, X. Li, and H. Zheng, “Design of Convolution Operation Accelerator based on FPGA,” in *Proc. Int. Conf. MLBDBI*, Taiyuan, China, 2020, pp. 80-84, doi:10.1109/MLBDBI51377.2020.00021.

[26] Y. LeCun, C. Cortes, and C. J. C. Burges, "The MNIST Database of handwritten digits," [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[27] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, “Regularization of Neural Networks using DropConnect,” in Proc. the International Conference on Machine Learning, PMLR, Atlanta, Georgia, USA, 2013, pp. 1058-1066.

[28] Siham Tabik, Ricardo F. Alvear-Sandoval, María M. Ruiz, José-Luis Sancho-Gómez, Aníbal R. Figueiras-Vidal, Francisco Herrera, “MNIST-NET10: A heterogeneous deep networks fusion based on the degree of certainty to reach 0.1% error rate. Ensembles overview and proposal,” *Information Fusion*, vol. 62, pp. 73-80, Oct. 2020. 10.1016/j.inffus.2020.04.002.

[29] Matuzas77, “MNIST-0.17,” [Online]. Available: https://github.com/Matuzas77/MNIST-0.17.

[30] S. S. Kadam, A. C. Adamuthe, and A. B. Patil, “CNN Model for Image Classification on MNIST and Fashion-MNIST Dataset,” *Journal of Scientific Research*, vol. 64, no. 2, pp. 374-384, 2020. 10.37398/JSR.2020.640251.

[31] R. F. Alvear-Sandoval, J. L. Sancho-Gómez, and A. R. Figueiras-Vidal, “On improving CNNs performance: The case of MNIST,” *Information Fusion*, vol. 52, pp. 106-109, Dec. 2019. 10.1016/j.inffus.2018.12.005.

[32] A. Baldominos, Y. Saez, and P. Isasi, “A Survey of Handwritten Character Recognition with MNIST and EMNIST,” *Appl. Sci.*, vol. 9, no. 15, 3169, Aug. 2019, 10.3390/app9153169.

[33] A. Velichko A, “Neural Network for Low-Memory IoT Devices and MNIST Image Recognition Using Kernels Based on Logistic Map,” *Electronics*, vol. 9, no. 9, 1432, 2020. 10.3390/electronics9091432.

[34] S. S, Mor, S. Solanki, S. Gupta, S. Dhingra, M. Jain, and R. Saxena, “Handwritten Text Recognition: with Deep Learning and Android,” *IJEAT*, vol. 8, no. 3S, pp. 819-825, Feb. 2019.

[35] S. Himanshu, “Activation Functions : Sigmoid, tanh, ReLU, Leaky ReLU, PReLU, ELU, Threshold ReLU and Softmax basics for Neural Networks and Deep Learning,” Jan. 19, 2019. [Online]. Available: https://himanshuxd.medium.com/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e.