ISSN No:-2456-2165

Desing of VLSI Architecture for a Flexible Test Bed of Artificial Neural Network for Training and Testing on FPGA

Gurmeet Kaur Arora
Electrical and Electronics communication Engineering
Indian Institute of Technology, Kharagpur Dewas, India

Abstract:- General-Purpose Processors (GPP)-based computers and Application Specific Integrated Circuits (ASICs) are the typical computing platforms used to develop the back propagation (BP) algorithm-based Artificial Neural Network (ANN) systems, but these computing devices constitute a hurdle for further advanced improvements due to a high requirement for sustaining a balance between performance and flexibility. In this work, architecture for BP learning algorithm using a 16-bit fixed- point representation is designed for the classification of handwritten digits on a field- programmable gate array (FPGA). The proposed design is directly coded and optimized for resource utilization and frequency in Verilog Hardware Description Language (HDL) and synthesized on the ML-605 Virtex 6 evaluation board. Experimental results show 10 times speedup and reduced hardware utilization when compared with existing implementations from literature. The architecture is expandable to other specifications in terms of number of layers, number of neurons in each layer, and the activation function for each neuron. The correctness of the proposed design is authenticated by comparing parameters obtained through Python code and Verilog.

***Keywords**:- General purpose processors (GPP), application specific integrated circuits (ASICs), artificial neural network (ANN), resource utilization, hardware descriptive language (HDL), field programmable gate-array (FPGA).*

I. INTRODUCTION

The biological neural networks that make up the human brain are deeply layered. These biological neural networks are able to recognize complicated things by first spotting basic traits and then combining them to pick up on complex ones. Similarly, an artificial neural network is trained to recognize different objects by first identifying small patterns inside the object and then integrating these simple patterns to identify complex patterns.

ML algorithms are generally complex and resource hungry thus implementation of Back propagation algorithm on low power device such as FPGA is much complicated than on GPUs or CPUs [18]. The data processing required obtaining the requisite convergence and accuracy by updating each weight makes BP computationally complex and time-consuming. Thus, almost all of the existing FPGA designs for ANNare based on software-hardware co-design [17] [11] [14] [19].

A tremendous amount of parallel computing operations are required by ANN architectures. Due to inherent parallelization and application-specific adaption, FPGAs are a realistic and affordable choice that, when compared to processor-based systems, helps in meeting the stringent speed requirements in real-time, delivers advanced AI services, and protects user privacy. Current ANN models emphasize a static and offline training mechanism in which the training data is pre-prepared. Nonetheless, training ANNs dynamically and adapting the models to the local environment is in great demand [16].

The goal is to provide a flexible testbed on FPGA where ML designers can specify their neural network architecture and fully or partially utilize the available hardware resources.

A multi-layered perceptron network of 784 x 32 x 10 is implemented and verified on the Xilinx Virtex 6 ML-605 evaluation board. Using 100 out of 60,000 training images and 100 out of 10,000 validation images from the MNIST dataset, this networkis trained and tested in Python and Verilog using the stochastic gradient descent BP algorithm, which has a learning rate of 0.125 and 10 epochs. The performance of Python and Verilog-based implementations is compared in terms of both accuracy and speed.

II. BACK PROPAGATION ALGORITHM

Back propagation is the process of calculating the error that is difference between the predicted output and the actual output and then propagating this error backwards through the network in order to revise the weights of the neurons. This is accomplished by calculating the gradient of the error with regard to the weights of each neuron using the chain rule of calculus.

$$e_{Ni} = y_{Ni} - d_{Ni} \qquad (1)$$

where, e_{Ni} is the error signal, d_{Ni} is the desired output and weight correction parameter can be summarized as

$$\begin{pmatrix}\text{Weight}\\ \text{correction}\end{pmatrix} = \begin{pmatrix}\text{learning-}\\ \text{rate parameter}\\ \eta\end{pmatrix} \times \begin{pmatrix}\text{local}\\ \text{gradient}\end{pmatrix} \times \begin{pmatrix}\text{input signal}\\ \text{of neuron } j\end{pmatrix}$$

III. METHODOLOGY

A. Finite State Machine for ANN

Each image undergoes six distinct phases during the computation process. The FSM used to execute back propagation in Verilog is illustrated in the figure 1. Table I provides a summary of each state and its corresponding function.

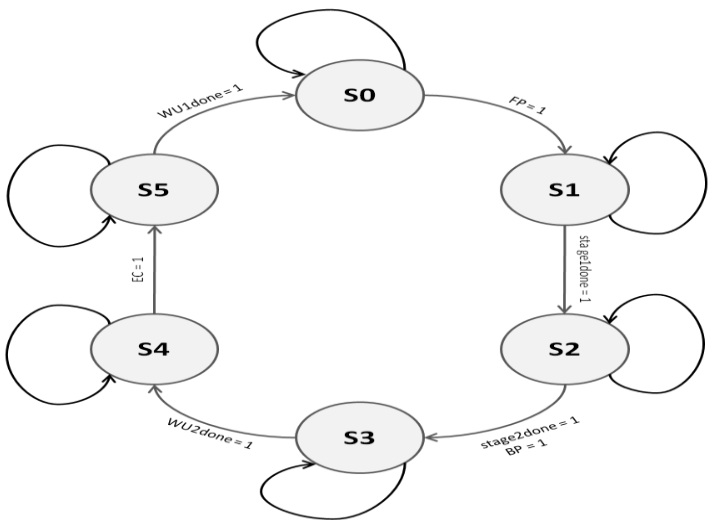


Fig. 1: FSM for ANN

Table 1: Summary of States and their functionality

State	Operation
S0	Load Input , Clear Registers
S1	Forward Propagation in hidden layer
S2	Forward Propagation in output layer
S3	Weight updation in output layer
S4	Error calculation for hidden layer
S5	Weight Updation in hidden layer

B. Multiply-Accumulate Unit

A Multiply-Accumulate (MAC) unit is an arithmetic logic unit (ALU) designed to perform two mathematical operations on two sets of input values: multiplication and addition. It multiplies two input values and then adds the product to an accumulator register, which contains the sum of all prior products computed by the MAC unit.

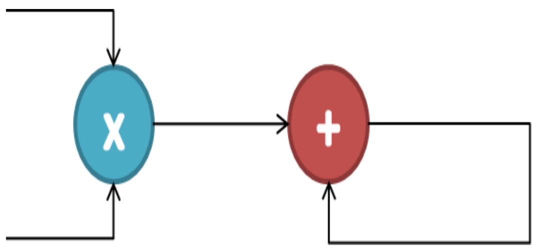


Fig. 2: MAC Unit

C. *Implementation of a Neuron*

A neuron is composed of a dedicated weight memory and a MAC unit, which is utilized based on the operational state. The register is capable of being loaded and its value depend on the operational state, where it is either 1'b0 in State 0, latched in States 1 and 2, and either latched or utilizing Weight2[counter] in State 3, 1'b0 in State 4, and utilizing either Weight1[counter] or latched in State 5. The value in State 3 and State 5 relies on the layer in which particular neuron belongs.

The output flow is determined by the Conditional Block, which either directs it to the LUT, the Weight memory, or the error computation block as shown in Figure 3. The Look up table (LUT) provides the activation value and its derivative with O(1) complexity when applied to the computed weighted sum using the magnitude of the weighted sum to address it. As there are no subsequent layers for the error signals to propagate back, neurons in the first layer do not utilize the error computation block. Multiplexers use the control signals generated by the FSM as their select lines.

IV. OVERALL DESIGN

A 784 x 32 x 10 architecture was constructed using the generate function in Verilog. The hidden layer has 32 neurons with a memory depth of 784, while the output layer has 10 neurons with a memory depth of 32 each. Some computations reuse MAC units while others require additional resources to achieve a balance between performance and resource utilization. The architecture shown in figure 4 operates as follows:

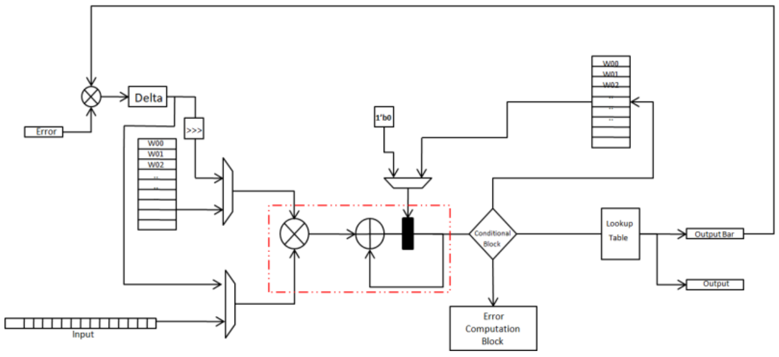


Fig. 3: Architecture of a Neuron

- In the first state, the MAC units of the first layer are active. The input pixels are multiplied by their respective weights, and the resulting weighted sum is passed through LUTs to derive H and Hbar simultaneously for all 32 neurons.
- In the second state, H and Hbars are sequentially multiplied by the output layer's weights, commencing from 0 to 31. The resulting weighted sum is then fed through LUTs to derive O and Obar for each of the 10 output neurons. At the end of this state, the error2 in the output layer is evaluated using 10 readily available subtractors, and delta2 value is determined using 10 multipliers. These operations are combinatorial and do not require any additional clock cycle.
- In the third state, output layer registers are serially loaded with weights from weight2 memory. This enables the learning rate, delta2, and H product to be added to the weight, and the weight is then written back to memory.
- To calculate the hidden layer error in state 4, multiply delta2 and Weights in sequence. Error1 for the present counter value is the sum of all of these partial products in a single cycle. To accomplish this, an adder with 10 operands is required. At the end of this state, delta1 values for all 32 neurons are calculated using the available error1 values.
- Within state 5, the input pixel is multiplied by the shifted delta1 value, and in one cycle, 32 weights in the hidden layer are updated. This step is similar to step 3 for weight updation of output layer.

A. *Clock Cycles*

For an arbitrary network (M x N x K x L) Clock cycles and number of computations can be generalized as follows: If we consider a network with dimensions 784 x 32 x 10, the number of cycles required or forward propagation is 816, which can be generalized as M+N+K. For back propagation, the number of cycles required is 848, which can be generalized as K + (K+N) + (N+M). Therefore, the total numberof cycles required is M+N+K+ K+ (N+K)+(N+M).

B. *Computational Units*

In a 784 x 32 x 10 network, there are 42 MAC units, 10 subtractors, one adder with ten operands, 42 multipliers, and 42 LUTs. If we generalise for a network with dimensions M x N x K x L, the required number of MAC units, multipliers, and LUTs would be N+K+L, the required number of subtractors would be L, and the required number of adders with N operands would be one, assuming N is greater than K and L.

The process of training and testing of MNIST dataset begins by loading the weight memory and input memory and initializing the values of parameters such as the number of hidden layers, neurons in each layer, activation function for each neuron, and number of epochs. The next step is to load an image to be trained and perform forward propagation. The weights are then updated, followed by an error calculation. This process is repeated from loading the image to be trained to weight updating 100 training images.

Once the training process is complete, the next step is to load an image to be tested and perform forward propagation. The output is then compared with the desired output, and if it matches, the value of accuracy is incremented. This process is repeated from loading the image to be tested to comparing the output with the desired output for 100 testing images. The whole process constitutes one epoch.

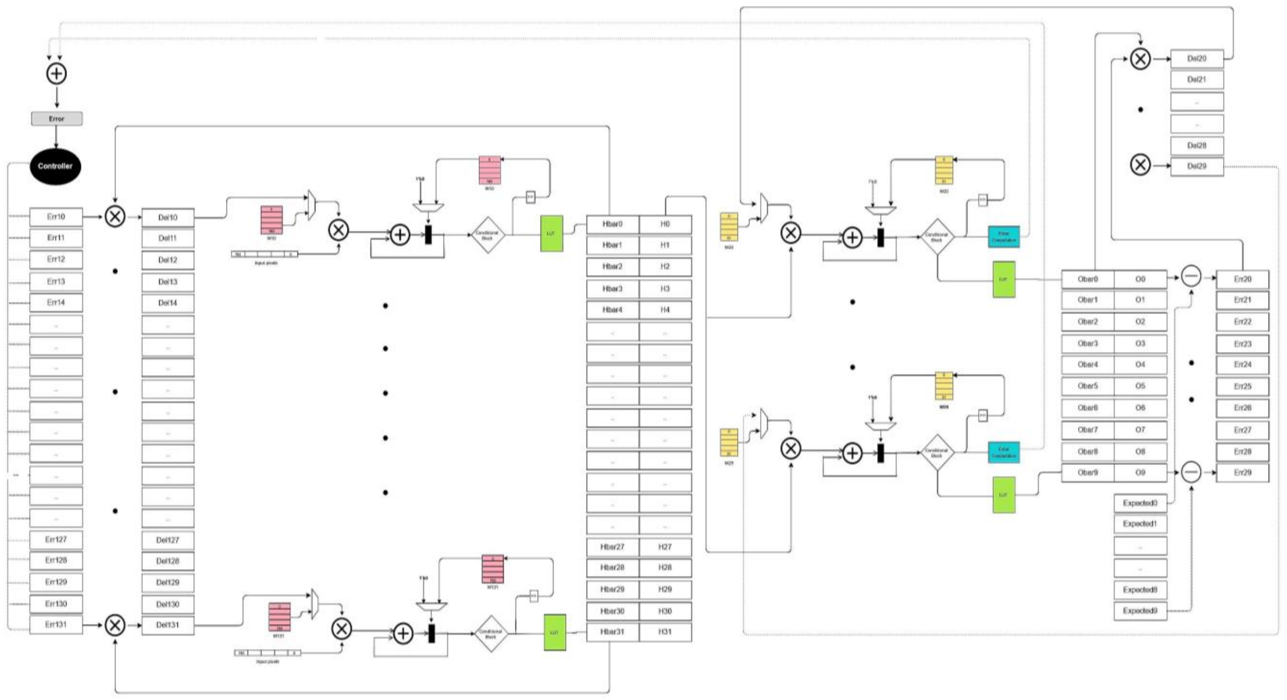


Fig. 4: ANN Architecture for 784 x 32 x 10 network for training and testing of MNIST Dataset

V. RESULTS AND COMPARATIVE ANALYSIS

Figure 5 shows the simulation results on Xilinx ISE 14.7 for the training and testing of the MNIST dataset for 100 images.

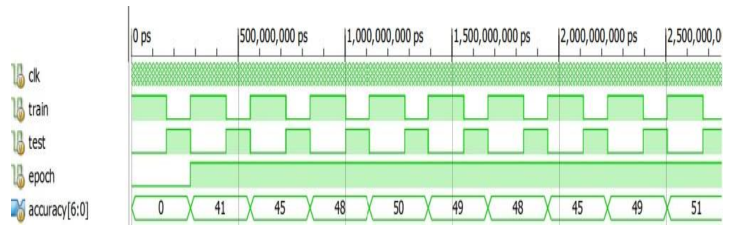
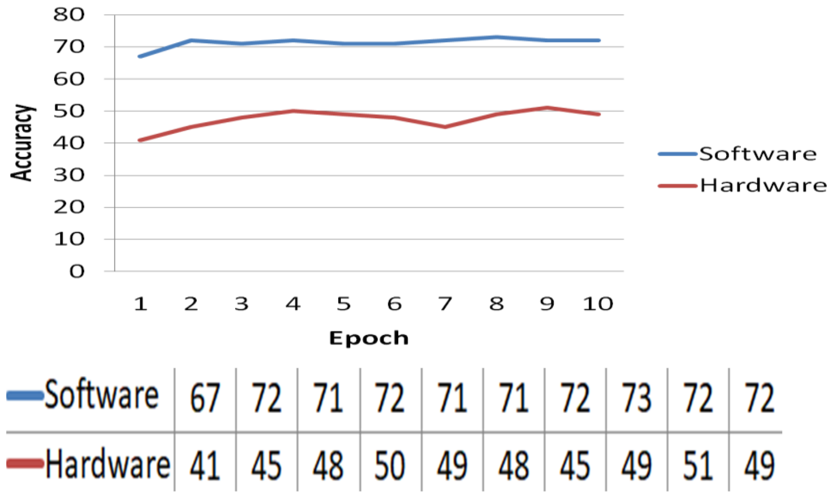


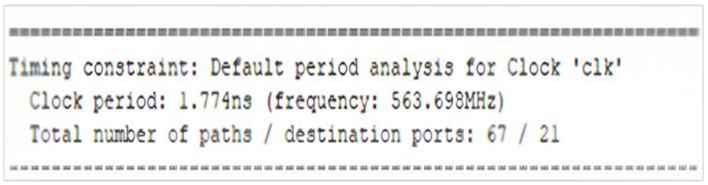
Fig. 5: Simulation results on Xilinx ISE 14.7 for training and testing of MNISTdataset for 100 images



Epoch	1	2	3	4	5	6	7	8	9	10
Software	67	72	71	72	71	71	72	73	72	72
Hardware	41	45	48	50	49	48	45	49	51	49

Fig. 6: Accuracy Comparison on different Platforms.

The Timing report obtained after synthesizing the design on Virtex 6 FPGA. The minimum clock period required for our design is 1.774 ns, and based on this value, we estimated the time required for one epoch to be 0.4403ms for 100 images. In contrast, the software-based implementation of the ANN requires approximately 4ms per epoch.



```
================================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 1.774ns (frequency: 563.698MHz)
  Total number of paths / destination ports: 67 / 21
================================================================================
```

The results of the timing analysis in hardware and software are tabulated in the table II, which shows that the hardware implementation is roughly 10 times faster than the software-based implementation. **Speedup achieved is 4/0.4403 = 9.08 or approximately 10.**

The proposed hardware-based implementation is approximately 10 times than the software-based implementation while sacrificing some accuracy. However, the results obtained show that the hardware-based implementation is a viable solution for applications where fast processing times are essential.

Table 2: Timing Comparison on different Platforms

Implementation	Time per Epoch
Software	Approx. 4ms
Hardware	0.4403 ms

Table III and table IV and presents a comparison between the proposed design and an existing implementation in terms of speed, resource and other parameters. The results indicate that the proposed design outperforms the existing design in terms of speed, flexibility, power and resource utilization.

Table 3: Comparison of Proposed design with High Level Synthesis based architectures from literature

Reference Papers	Comparison Criteria	Target	Achieved	Comparative Advantage / Observation
High-Level Synthesize of Back propagation Artificial Neural Network Algorithm on the FPGA **[1]**	Speed	106.72 MHz (Network : 2 x 2 x 1)	563.698 MHz	• Optimized • High Speed
Back propagation Algorithm and its Hardware Implementations: A Review **[8]**				• Optimized
FPGA Implementation of Handwritten Number Recognition using Artificial Neural Network **[11]**	1. Number Representation 2. Role of FPGA	1. Floating Point Representation 2. Inference Machine	1. Fixed Point Representation. 2. Complete Machine	• Scalable • Resource Efficient • Simple • Independent and Flexible

Furthermore, it is worth noting that the proposed design is more efficient than the existing design, which only implements the forward propagation part in hardware as shown in table V.

Table 4: Comparison of Proposed design with RTL designed based architectures fromliterature

Reference Papers	Comparison Criteria	Target	Achieved	Comparative Advantage / Observation
1. A systolic array exploiting the inherent parallelism of ANN **[4]** 2. Digital VLSI Architecture for Neural Networks **[7]**	Architectural Component Used	Systolic Array Multiplier	MAC Units	• Energy Efficient • Simpler design • MAC units processes data sequentially
Implementation of Back Propagation Algorithm in Verilog **[12]**				Our Design is inspired by the neuron implemented in this reference paper
Implementing the on-chip back propagation learning algorithm on FPGA architecture **[16]**	Number Representation	32-bit Floating Point Representation (Network : 4 x 4 x 1)	16-bit Fixed Point Representation	• Resource Efficient • Simple • Deterministic
Robust Handwritten Digit Recognition System using Hybrid Artificial Neural Network on FPGA **[14]**	1. Number Representation 2. Role of FPGA	1. Fixed Point Representation 2. Inference Machine	1. Fixed Point Representation 2. Complete Machine	• Independent • Scalable • Flexible
Design of Artificial Neural Network Architecture for Handwritten Digit Recognition on FPGA **[17]**	1. Number Representation 2. Frequency 3. Method 4. Role of FPGA	1. 16-bit Floating point 2. 205 MHz 3. PCA (20 x 12 x 10) 4. Inference Machine	1. 16-bit Fixed point 2. 563.698 MHz 3. No PCA 4. Complete Machine	• Deterministic • Independent • High Speed • No pre-processing of data is required • Simple and Efficient
FPGA acceleration on multi-layer perceptron neural network for digit recognition **[18]**	1. Role of FPGA 2. Time per image 3. Resource Utilization	1. Inference Machine (784 x 12 x 10) 2. 1.55 μs for FP	1. Complete Machine 2. 1.412 μs for FP	• Similar Implementation • Resource Friendly • Flexible • Scalable • Fast

Table 5: Comparison of Hardware requirements for forward propagation

Design Structure	Latency (Cycles)	Hardware Cost					
		MUL	ADD	SUB	EXP	REC	LUT
Non -Pipelined	26	9528	9528	44	22	22	0
Fully - Pipelined	48	796	796	4	2	2	0
8 – Stage Pipelined	129	110	110	4	2	2	0
Proposed Design	795	22	22	NIL	NIL	NIL	22

It shows that roughly 0.2%, 2.76%, 20% of the hardware of Non-pipelined, Fully-pipelined and 8-stage pipelined, respectively is utilized in our design when compared with architecture 784 x 12 x 10 of reference [19].

VI. CONCLUSION

This work presents a novel FPGA-based implementation of an artificial neural network that offers reconfigurability in terms of the number of layers, neurons, and activation functions for each layer. The implementation provides faster computation speed than software-based implementation, but accuracy is compromised due to fixed-point computation. The design also outperforms pre- existing hardware-based implementations in terms of frequency and resource utilization. This work represents a significant contribution in demonstrating the potential of FPGA-based implementation in accelerating neural network prediction, and not just limiting the use of FPGA for recognition phase.

Future work for this research includes incorporating a linear feedback shift register (LFSR) to generate random numbers for weight initialization, processing real-time data using serial communication techniques, and improving the accuracy of the hardware-based implementation.

It should be noted that almost all the recent research papers have focused on ANN inference with FPGAs, training an ANN with FPGAs has not been well exploited by the community. This is likely due to the complexity of designing an FPGA system that can effectively pipeline the processes . The flexibility of FPGAs in terms of integrated circuit reconfiguration provides opportunities for implementing a wide range of operations and instructions.

Future research in this area can explore more complex hardware architectures and improved number representations to enhance the accuracy of hardware-based implementations.

REFERENCES

[1.] Nuzula Afianah, Agfianto Eko Putra, and Andi Dharmawan. High-level synthesize of backpropagation artificial neural network algorithm on the fpga. In *2019 5th International conference on science and technology (ICST)*, volume 1, pages 1–5. IEEE, 2019.

[2.] Ramón J Aliaga, Rafael Gadea, Ricardo J Colom, José M Monzó, Christoph W Lerche, and Jorge D Mart´ınez. System-on-chip implementa- tion of neural network training on fpga. *International Journal On Advances in Systems and Measurements Volume 2, Number 1, 2009*, 2009.

[3.] Song Bo, Kensuke Kawakami, Koji Nakano, and Yasuaki Ito. An rsa en- cryption hardware algorithm using a single dsp block and a single block ram on the fpga. *International Journal of Networking and Computing*, 1(2):277–289, 2011.

[4.] Mohammadreza Esmali Nojehdeh, Levent Aksoy, and Mustafa Altun. Effi- cient hardware implementation of artificial neural networks using approxi-mate multiply-accumulate blocks. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 96–101, 2020.

[5.] Simon S. Haykin. *Neural networks and learning machines*. Pearson Edu- cation, Upper Saddle River, NJ, third edition, 2009.

[6.] Albert Knebel and Dorin Patru. Educational neural network development and simulation platform. In *2020 St. Lawrence Section Meeting*, 2020.

[7.] S.Y. Kung and J.N. Hwang. Digital vlsi architectures for neural networks. In *IEEE International Symposium on Circuits and Systems,*, pages 445–448 vol.1, 1989.

[8.] Shivani Kuninti and S Rooban. Backpropagation algorithm and its hard- ware implementations: A review. In *Journal of Physics: Conference Series*, volume 1804, page 012169. IOP Publishing, 2021.

[9.] Yihua Liao. Neural networks in hardware: A survey. *Department of Com- puter Science, University of California*, 2001.

[10.] Sainath Shravan Lingala, Swanand Bedekar, Piyush Tyagi, Purba Saha, and Priti Shahane. Fpga based implementation of neural network. In *2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, pages 1–5. IEEE, 2022.

[11.] Harsh Mittal, Abhishek Sharma, and Thinagaran Perumal. Fpga imple- mentation of handwritten number recognition using artificial neural net- work. In *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, pages 1010–1011. IEEE, 2019.

[12.] Esraa Z. Mohammed and Haitham Kareem Ali. Hardware implementationof artificial neural network using field programmable gate array. *Interna- tional Journal of Computer Theory and Engineering*, pages 780–783, 2013.

[13.] Syahrulanuar Ngah, Rohani Abu Bakar, Abdullah Embong, and Saifudin Razali. Two-steps implementation of sigmoid function for artificial neural network in field programmable gate array. *ARPN journal of engineering and applied sciences*, 11(7):4882–4888, 2016.

[14.] R Pramodhini, Sunil S. Harakannanavar, CN Akshay, N Rakshith, Ritwik Shrivastava, and Akchhansh Gupta. Robust handwritten digit recogni- tion system using hybrid artificial neural network on fpga. In *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)*, pages 1–5,2022.

[15.] L Ranganath, D Jay Kumar, and P Siva Nagendra Reddy. Design of mac unit in artificial neural network architecture using verilog hdl. In *2016 In- ternational Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 607–612. IEEE, 2016.

[16.] Yudong Tao, Rui Ma, Mei-Ling Shyu, and Shu-Ching Chen. Chal- lenges in energy-efficient deep neural network training with fpga. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recogni- tion Workshops (CVPRW)*, pages 1602–1611, 2020.

[17.] Huynh Viet Thang. Design of artificial neural network architecture for handwritten digit recognition on fpga. *Tp chí Khoa hc và Công ngh-i hc à Nng*, 2016.

[18.] Huan Minh Vo. Implementing the on-chip back propagation learning algo- rithm on fpga architecture. In *2017 International Conference on System Science and Engineering (ICSSE)*, pages 538–541, 2017.

[19.] Isaac Westby, Xiaokun Yang, Tao Liu, and Hailu Xu. Fpga acceleration on a multi-layer perceptron neural network for digit recognition. *The Journal of Supercomputing*, 77(12):14356–14373, 2021.

.