https://journal.utripoli.edu.ly/index.php/Alqalam/index eISSN 2707-7179

*Original article*

# Implementation of Artificial Neural Networks on Field-Programmable Gate Arrays

**Salim Adrees[1]*, Ala Abdulrazeg[1], Wafa Shuaieb[2]**

*[1]Department of Computer Engineering, Faculty of Engineering, Omar Al-Mukhtar University, Al-Bayda, Libya*
*[2]Department of Electrical & Electronic Engineering, Faculty of Engineering, Omar Al-Mukhtar University, Al-Bayda, Libya*

## ARTICLE INFO

***Corresponding Email.*** Salim.ali@omu.edu.ly

***Received***: 01-08-2024
***Accepted***: 29-09-2024
***Published***: 02-10-2024

***Keywords***. *Neural Network, FPGA, Artificial Intelligent, Verilog, Hardware Design.*

## ABSTRACT

*Implementing Artificial Neural Networks (ANNs) on Field-Programmable Gate Arrays (FPGAs) provides a promising solution for achieving high-performance, low-latency, and energy-efficient computations in complex tasks. This paper investigates the methodology for mapping ANNs onto FPGAs, focusing on critical aspects such as architecture selection, hardware design, and optimization techniques. By harnessing the parallel processing capabilities and reconfigurability of FPGAs, neural network computations are significantly accelerated, making them ideal for real-time applications like image processing and embedded systems. The implementation process addresses key considerations, including fixed-point arithmetic, memory management, and dataflow optimization, while employing advanced techniques such as pipelining, quantization, and pruning. The research compares the accuracy and performance speedup of ANNs on CPUs versus FPGAs, revealing that FPGA-based simulations are 4680 times faster than CPU-based simulations using MATLAB, without compromising prediction accuracy.*

***Cite this article.*** *Adrees S, Abdulrazeg A, Shuaieb W. Implementation of Artificial Neural Networks on Field-Programmable Gate Arrays. Alq J Med App Sci. 2024;7(4):927-933.* https://doi.org/10.54361/ajmas.247405

## INTRODUCTION

Artificial intelligent is now a driving force behind various technologies used in websites, cameras, and smartphones. It is often implemented to identify objects in images and extract them for further processing. The general approach of machine learning involves taking a real dataset and applying algorithms such as deep learning [1], neural networks [2], the Perceptron algorithm [3], K-nearest neighbor [4], decision trees [5], among others. Among these, ANN has emerged as one of the most dominant techniques. ANNs excel in extracting and analyzing data, allowing them to effectively establish relationships between inputs and outputs.

### *Artificial Neural Networks*

Artificial Neural Networks (ANNs) have emerged as a pivotal technology in the field of artificial intelligence (AI), driving advancements across numerous domains including computer vision, natural language processing, and autonomous systems. Inspired by the intricate networks of neurons in the human brain, ANNs are designed to simulate the way biological systems process information, offering a powerful framework for addressing complex computational tasks that traditional algorithms struggle to handle [6-8].
The resurgence of interest in ANNs over the past decade can be attributed to several key developments. First, the exponential growth in computational power, facilitated by the advent of Graphics Processing Units (GPUs) and distributed computing, has made it feasible to train large-scale neural networks. Second, the availability of massive

datasets, driven by the proliferation of digital data in various sectors, has provided the necessary fuel for training deep neural networks. Finally, innovations in learning algorithms, particularly the backpropagation algorithm and its variants, have significantly improved the training efficiency and generalization capabilities of ANNs [9].

## *Structure of an ANN*

A General Artificial Neural Network (ANN) is a computational model inspired by the structure and function of the human brain. It consists of interconnected layers of nodes (neurons), which work together to process information and make predictions or decisions. Figure 1 shows the general ANN model architecture.

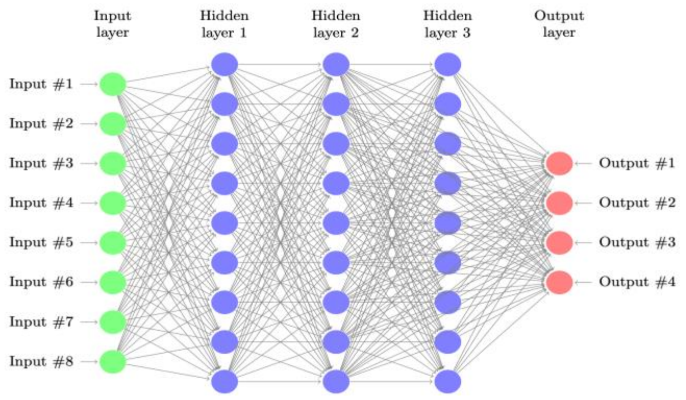


***Figure 1. General ANN Model architecture***

- Input Layer: This layer receives the input data. Each neuron in this layer represents a feature or attribute of the input data.
- Hidden Layers: These layers process the inputs received from the input layer. An ANN can have one or multiple hidden layers, and the neurons in these layers apply various transformations to the data using activation functions.
- Output Layer: This layer produces the final output. The number of neurons in the output layer depends on the classes of the application.

## *Software implementation*

The application which will be addressed in this paper is digit recognition. The dataset is self-made binary digit dataset. The number of classes are 5 classes (1 : 5) which are represented as shown in Figure 2.

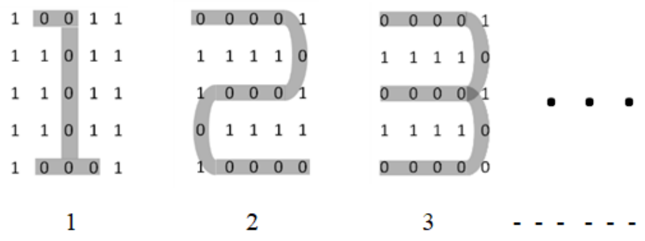


***Figure 2. Dataset description***

Every input image is (5 x 5) binary array, every bit will be an input of each neuron in the input layer, the image will be reshaped to be (25 x 1). A fully connected 5 layers ANN will be implemented in this application as shown in Figure 3.

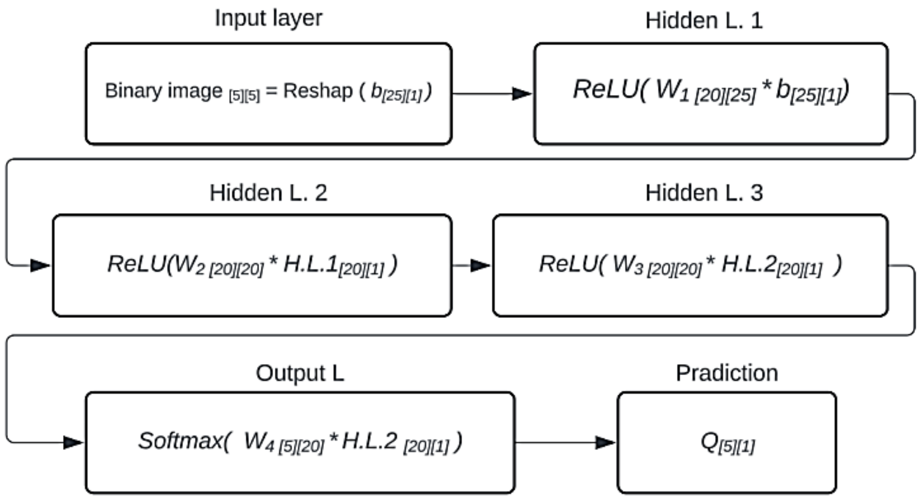

Input layer
Binary image [5][5] = Reshap ( b[25][1] )
Hidden L. 1
ReLU( W1 [20][25] * b[25][1])
Hidden L. 2
ReLU(W2 [20][20] * H.L.1[20][1] )
Hidden L. 3
ReLU( W3 [20][20] * H.L.2[20][1] )
Output L
Softmax( W4 [5][20] * H.L.2 [20][1] )
Pradiction
Q[5][1]

***Figure 3. Flowchart of the software implementation of ANN***

The first layer is an input layer which consist of 25 inputs corresponding to the number of bits in the binary image. The second layer is the first hidden layer consist of 20 neurons with Rectified Linear Unit (ReLU) activation function. The corresponding equation in this layer is represented in Equation 1.

$$\boldsymbol{Hidden\ L.1_{[20][1]} = ReLU\left(W_{1_{[20][25]}} * b_{[25][1]}\right)} \qquad \textbf{Equation 1}$$

The ReLU activation function is one of the most popular in neural networks. ReLU is defined as being zero for all inputs below a certain threshold (usually zero) and linear for inputs above that threshold. The function can be mathematically expressed in Equation 2[10].

$$\boldsymbol{ReLU(x) = max(0, x)} \qquad \textbf{Equation 2}$$

The second and the third hidden layers consist of 20 neurons and follow the same mathematics as the first layer. Equation 3 and Equation 4 represent the mathematical equations of layer 2 and layer 3 respectively.

$$\boldsymbol{Hidden\ L.2_{[20][1]} = ReLU\left(W_{2_{[20][20]}} * Hidden\ L.1_{[20][1]}\right)} \qquad \textbf{Equation 3}$$

$$\boldsymbol{Hidden\ L.3_{[20][1]} = ReLU\left(W_{3_{[20][20]}} * Hidden\ L.2_{[20][1]}\right)} \qquad \textbf{Equation 4}$$

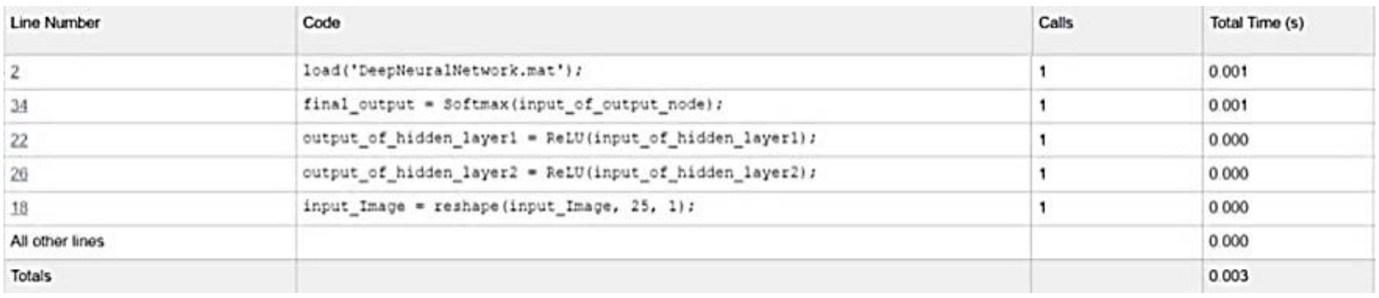
The final layer is the output layer consists of 5 neurons, The corresponding equation in this layer is represented in Equation 5.

$$\boldsymbol{Output\ L._{[5][1]} = Softmax\left(W_{4_{[5][20]}} * Hidden\ L.3_{[20][1]}\right)} \qquad \textbf{Equation 5}$$

Softmax is commonly used in the output layer of a classifier when dealing with more than two categories. The function can be mathematically expressed in Equation 6 [11].

$$\boldsymbol{Sotmax\ (z)_i = {e^{z_i}}\Big/{\sum_{j=1}^{k} e^{z_j}}} \qquad \textbf{Equation 6}$$

Converting the following equation to MATLAB code, train the module to find the evaluation characteristics and then test the module to find the validation characteristics and finally apply profiling function to find the execution time in CPU. The results show that evaluation accuracy of the module reached 96% and validation accuracy of the module reached 92%. Figure 4 shows the execution time in MATLAB, and the results show that it took 0.003 s to find the output of a given image.



| Line Number | Code | Calls | Total Time (s) |
|---|---|---|---|
| 2 | `load('DeepNeuralNetwork.mat');` | 1 | 0.001 |
| 34 | `final_output = Softmax(input_of_output_node);` | 1 | 0.001 |
| 22 | `output_of_hidden_layer1 = ReLU(input_of_hidden_layer1);` | 1 | 0.000 |
| 26 | `output_of_hidden_layer2 = ReLU(input_of_hidden_layer2);` | 1 | 0.000 |
| 18 | `input_Image = reshape(input_Image, 25, 1);` | 1 | 0.000 |
| All other lines | | | 0.000 |
| Totals | | | 0.003 |

*Figure 4. Execution time in MATLAB*

### *Field Programmable Gate Arrays*

Field Programmable Gate Arrays (FPGAs) represent a versatile and powerful technology in digital design and embedded systems. Since their introduction in the mid-1980s, FPGAs have become essential components across diverse applications, including telecommunications, automotive systems, high-performance computing, and artificial intelligence. Unlike Application-Specific Integrated Circuits (ASICs), which are tailored for a single function, FPGAs offer reconfigurability, enabling designers to modify hardware configurations even after manufacturing. This adaptability, combined with their parallel processing capabilities, makes FPGAs invaluable for creating custom hardware solutions tailored to specific needs.

The growing complexity of digital systems, alongside increasing demands for higher performance and lower power consumption, has accelerated the adoption of FPGAs across various industries. Their reconfigurability allows for rapid prototyping, iterative design, and the implementation of complex algorithms directly in hardware, circumventing some limitations of traditional software approaches. Moreover, FPGAs are increasingly utilized in areas like deep learning acceleration, real-time data processing, and edge computing, where they deliver significant advantages over general-purpose processors in terms of latency, throughput, and energy efficiency.[12-14].

### *FPGA implementation*

Hardware design of the module is implemented in Vivado Design Suite. it is a comprehensive software suite for the synthesis and analysis of hardware description language (HDL) designs. It supersedes Xilinx ISE, offering enhanced features for system-on-chip (SoC) development and high-level synthesis. Unlike its predecessor, Vivado represents a complete ground-up rewrite and rethinking of the entire design flow, providing a more efficient and integrated environment for modern hardware design. The programming language used in Vivado is Verilog HLS. The hardware design will be able to accept a 25 bits binary image and predict the correct digit, which means that the testing algorithm will be converted to hardware design. After train the module in MATLAB, the learned weights will be exported to Vivado. The design will be implemented with behavioral modeling style. The flowchart is shown in Figure 5.

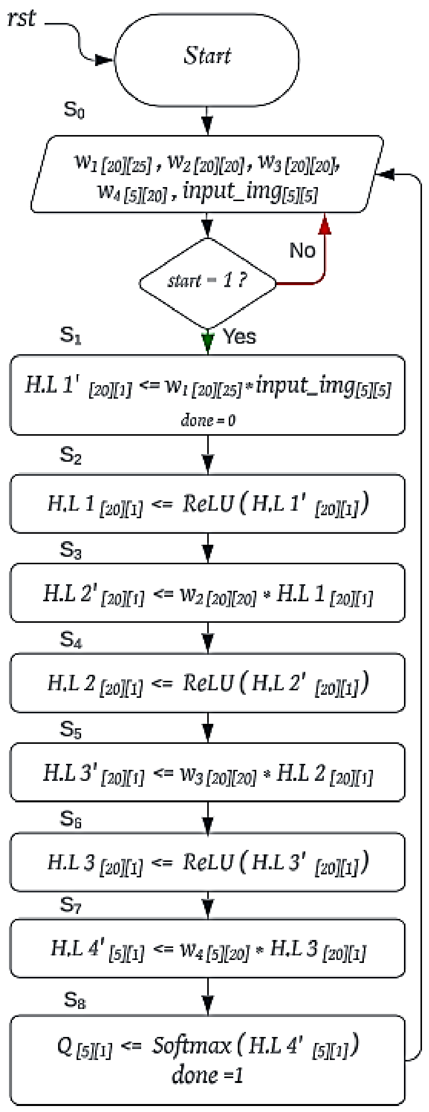
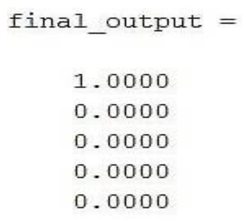

rst
Start
S0
w1 [20][25] , w2 [20][20] , w3 [20][20], w4 [5][20] , input_img[5][5]
start = 1 ?
No
S1
Yes
H.L 1' [20][1] <= w1 [20][25] * input_img[5][5]
done = 0
S2
H.L 1 [20][1] <= ReLU ( H.L 1' [20][1])
S3
H.L 2' [20][1] <= w2 [20][20] * H.L 1 [20][1]
S4
H.L 2 [20][1] <= ReLU ( H.L 2' [20][1])
S5
H.L 3' [20][1] <= w3 [20][20] * H.L 2 [20][1]
S6
H.L 3 [20][1] <= ReLU ( H.L 3' [20][1])
S7
H.L 4' [5][1] <= w4 [5][20] * H.L 3 [20][1]
S8
Q [5][1] <= Softmax ( H.L 4' [5][1])
done =1

***Figure 5. Flowchart of Hardware Implementation***

The design operates at a clock frequency of 100 MHz, which corresponds to a 20 ns clock cycle. Behavioral modeling, the highest level of abstraction in Verilog, has been employed for this project. This approach enables the implementation of the module based on the desired design algorithm without focusing on specific hardware details like the number of adders, multipliers, or dividers needed. The Vivado software tool was used to create an Algorithmic State Machine (ASM) chart for the entire design. The module has been successfully synthesized in Vivado and is now ready for testing with a 5x5 binary image. MATLAB was used to test the module, and the output predictions, shown in Figure 6, indicate that the module correctly identifies the digit "1" in the image



```
final_output =

    1.0000
    0.0000
    0.0000
    0.0000
    0.0000
```

***Figure 6. Results of a given image***

By taking the same image and implement it in Vivado design suite to test the module and confirm the accuracy of the hardware design, the result is shown in Figure 7.

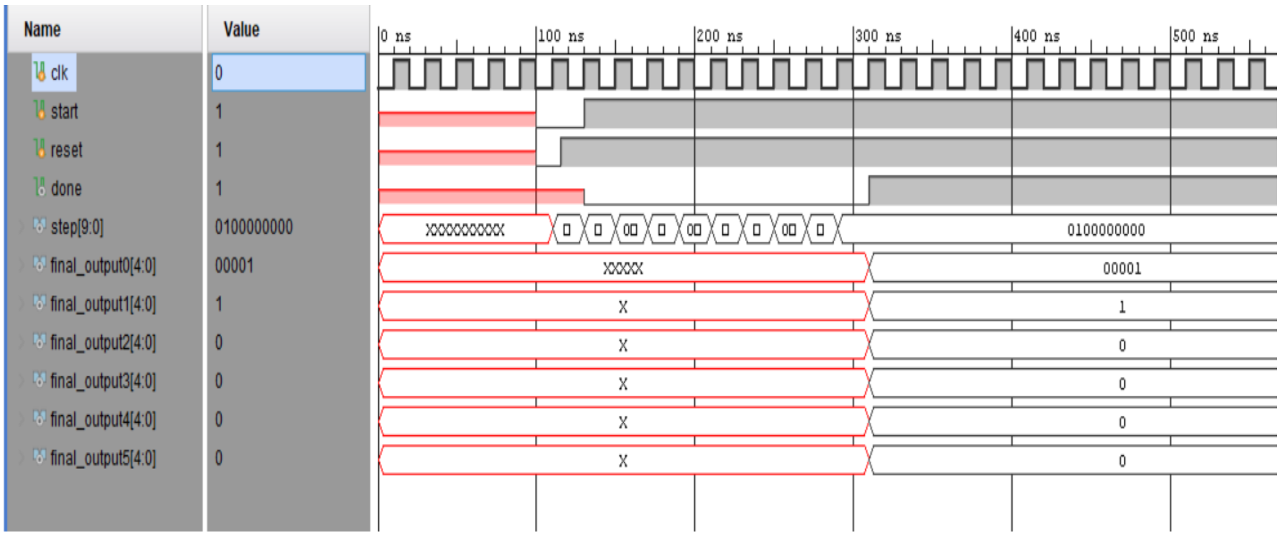


*Figure 7. Results of the hardware implementation*

The results indicate that the outputs from MATLAB Figure 6 and the Vivado Design Suite Figure 7 are identical, confirming that the hardware implementation has been correctly designed and synthesized. Additionally, when testing the module with five different binary images, the outputs in both MATLAB and Vivado were consistent, further validating the design's accuracy.

The second aspect of the evaluation focused on execution time. The timing simulation in the Vivado Design Suite revealed that the output delay is 200 ns, corresponding to 10 clock cycles. This demonstrates a significant improvement in timing performance. Table 1 compares the timing simulation between the CPU and FPGA, showing that the FPGA simulation is 4680 times faster than the CPU simulation.

*Table 1. comparison between CPU and FPGA timing simulation*

| Timing | CPU | FPGA |
|---|---|---|
| | 0.003 s (150000 clock cycles) | 200 ns (10 clock cycles) |

## CONCLUSION

One of the most challenging aspects of this paper is exporting inputs from the MATLAB implementation due to the complexity of the code. Additionally, working with fixed-point numbers is generally easier and more efficient than using floating-point numbers. Implementing floating-point arithmetic in hardware requires IP cores, which can consume many clock cycles. However, using fixed-point numbers does not significantly impact the results, as the introduced error is negligible. The FPGA module was designed using behavioral modeling, a style that abstracts away specific hardware details like the number of adders, multipliers, and registers. The software tool automatically determines these resources, which is a key advantage of this modeling approach. However, a significant drawback is that behavioral designs can be challenging for the software tool to synthesize and translate into a hardware design. If the module is highly complex, behavioral modeling may not be the most suitable choice. Finally, in real-time applications, execution time is critical. For certain algorithms, implementing the most critical parts in an FPGA can minimize output delay without compromising the accuracy of the results.

***Conflict of interest***. Nil

## REFERENCES

1. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436-44; doi: 10.1038/nature14539.
2. Muthuramalingam A, Himavathi S, Srinivasan E. Neural Network Implementation Using FPGA: Issues and Application. Int J Inform Technol. 2007;4.
3. Taright Y, Hubin M. FPGA implementation of a multilayer perceptron neural network using VHDL. ICSP '98 1998 Fourth International Conference on Signal Processing (Cat No98TH8344). 1998;2:1311-4 vol.2.
4. Li Z-H, Jin J, Zhou X-g, Feng Z-H. K-nearest neighbor algorithm implementation on FPGA using high level synthesis. 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT). 2016:600-2.

5. Narayanan R, Honbo D, Memik G, Choudhary A, Zambreno J: Interactive presentation: An FPGA implementation of decision tree classification. In: Proceedings of the conference on Design, automation and test in Europe. Nice, France: EDA Consortium; 2007: 189–94.
6. Agatonovic-Kustrin S, Beresford R. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. Journal of pharmaceutical and biomedical analysis. 2000;22(5):717-27.
7. Lillicrap TP, Santoro A, Marris L, Akerman CJ, Hinton G. Backpropagation and the brain. Nature Reviews Neuroscience. 2020;21(6):335-46.
8. Hecht-Nielsen R. Theory of the backpropagation neural network. In: Neural networks for perception. Elsevier; 1992. p. 65-93.
9. Talib MA, Majzoub S, Nasir Q, Jamal D. A systematic literature review on hardware implementation of artificial intelligence algorithms. The Journal of Supercomputing. 2021;77(2):1897-938.
10. Bai Y: RELU-function and derived function review. In: SHS Web of Conferences. vol. 144: EDP Sciences; 2022: 02006.
11. Kouretas I, Paliouras V: Simplified hardware implementation of the softmax activation function. In: 2019 8th international conference on modern circuits and systems technologies (MOCAST). IEEE; 2019: 1-4.
12. Wang C, Luo Z. A review of the optimal design of neural networks based on FPGA. Applied Sciences. 2022;12(21):10771.
13. Yang H, Zhang J, Sun J, Yu L. Review of advanced FPGA architectures and technologies. Journal of Electronics (China). 2014;31(5):371-93.
14. Adrees SA, Abdulrazeg AA. Capsule Network Implementation On FPGA. Journal of Pure & Applied Sciences. 2020;19(5):50-4.

## مصفوفات البوابات القابلة للبرمجة

**سالم ادريس[1]*، علاء عبد الرازق[1]، وفاء شعيب[2]**

[1]قسم هندسة الحاسوب، كلية الهندسة، جامعة عمر المختار، البيضاء، ليبيا

[2]قسم الهندسة الكهربائية والإلكترونية، كلية الهندسة، جامعة عمر المختار، البيضاء، ليبيا

**المستخلص**

يوفر تطبيق الشبكات العصبية الاصطناعية على مصفوفات البوابات القابلة للبرمجة ميدانيًا حلاً واعدًا لتحقيق عمليات حسابية عالية الأداء و زمن أقل وكفاءة في استخدام الطاقة في المهام المعقدة. يتحقق هذا البحث من منهجية تطبيق الشبكات العصبية الاصطناعية على مصفوفات البوابات القابلة للبرمجة ميدانيًا، مع التركيز على الجوانب الحرجة مثل اختيار ألية التصميم وتصميم الدائرة الإلكترونية وتقنيات التحسين. من خلال الاستفادة من قدرات المعالجة المتوازية وإعادة برمجة مصفوفات البوابات القابلة للبرمجة ميدانيًا، يتم تسريع عمليات حساب الشبكة العصبية بشكل كبير، مما يجعلها مثالية للتطبيقات التي تتطلب الاستجابة اللحظية مثل معالجة الصور والأنظمة المضمنة. عملية التنفيذ تشمل بعض المفاهيم الاساسية، بما في ذلك حساب الاعداد الكسرية، وإدارة الذاكرة، وتحسين تدفق البيانات، مع استخدام تقنيات متقدمة مثل تنفيذ اكثر من عملية في وقت واحد وتقليل الوحدات الالكترونية المستخدمة. يقارن البحث بين دقة وسرعة أداء خوارزمية الشبكات العصبية الاصطناعية على وحدات المعالجة المركزية مقابل وحدات مصفوفات البوابات القابلة للبرمجة ميدانيًا ، حيث وجد أن تطبيق الخوارزمية على مصفوفات البوابات القابلة للبرمجة ميدانيًا أسرع بمقدار 4680 مرة من تطبيق الخوارزمية على وحدة المعالجة المركزية باستخدام MATLAB ، مع المحافظة على دقة المخرجات.

**الكلمات المفتاحية**. الشبكة العصبية، مصفوفات البوابات القابلة للبرمجة ميدانيًا ، الذكاء الاصطناعي، Verilog، التصميم الالكتروني.