Vol. 34, No. 02, pp. 807-824/Junio 2021

ISSN-E 1995-9516
Universidad Nacional de Ingeniería
COPYRIGHT © (UNI). TODOS LOS DERECHOS RESERVADOS
http://revistas.uni.edu.ni/index.php/Nexo
https://doi.org/10.5377/nexo.v34i02.11568

An efficient FPGA implementation of hand gestures recognition based on neural networks

Una eficiente implementación en FPGA del reconocimiento de gestos de mano basado en redes neurales

Ali Abdolazimi[1], Amir Sabbagh Molahosseini[1,*], Farshid Keynia [2]

[1] Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran.
[2] Department of Energy Management and Optimization, Institute of Science and High Technology and Environmental Sciences, Kerman, Iran.

[*] amir@iauk.ac.ir

(recibido/received: 12-enero-2021; aceptado/accepted: 08-abril-2021)

ABSTRACT

Different gestures of hand which is a powerful communication channel between man to man and/or man to machine transfers a large amount of information in our daily lives. For example, sign languages are widely used by individuals with speech handicaps. Recognizing hand gestures in the image can be considered a powerful parameter in man-to-machine communication. Although researchers have been trying to implement different hand gestures on several hardware platforms over the past years, their attempts have been confronted by many challenges including restricted resources of hardware platforms, noise factors in the environment, or insufficient accuracy of output in high numbers of experimental samples. In this work, an optimum and parallelized method is developed to implement recognition of different hand gestures in the image on FPGA. The introduced method uses an MLP network with high numbers of hidden layers without wasting resources of the hardware platform. The results comparing the proposed optimized method with the state-of-the-art methods show that the suggested method can be implemented on an FPGA platform with high output accuracy and lower resources.

Keywords: MLP, FPGA implementation, Hand gestures recognition, Fourier transform, Neural network.

RESUMEN

Los diferentes gestos de la mano que es un poderoso canal de comunicación entre hombre a hombre y / o hombre a máquina, transfieren gran cantidad de información en nuestra vida diaria. Por ejemplo, los lenguajes de señas son ampliamente utilizados por personas con discapacidad del habla. El reconocimiento de los gestos de las manos en la imagen puede considerarse como un parámetro poderoso en la comunicación hombre-máquina. Aunque los investigadores han intentado implementar diferentes gestos con las manos en varias plataformas de hardware durante los últimos años, sus intentos se han enfrentado a muchos desafíos, incluidos los recursos restringidos de las plataformas de hardware, factores de ruido en el entorno o una precisión insuficiente de la salida en un gran número de muestras experimentales. En este trabajo se desarrolla un método óptimo y paralelizado para implementar el reconocimiento de diferentes gestos con las manos en imagen en FPGA. El método introducido utiliza una red MLP con un gran número de capas ocultas sin desperdiciar recursos de la plataforma de hardware. Los resultados que comparan el método optimizado propuesto con los métodos de última generación muestran que el método sugerido se puede implementar en la plataforma FPGA con una alta precisión de salida y menos recursos.

Palabras claves: MLP, Implementación FPGA, Reconocimiento de gestos con las manos, Transformada de Fourier, Red neuronal.

1. INTRODUCTION

Hand movements are a simple and natural way of interacting. Even, people who can speak, usually make use of many different movements to help their communications. Movements of hand can be used in a wide range of the current applied programs through movements (different sign languages) and applying objects in Virtual Environment (VE) as a way of an interacting man with the computer. To achieve a natural and comprehensive interaction between man and computer, a man's hand can be used as a connecting device (Kirishima, Sato, & Chihara, 2005). From years ago, fast information processing was a matter of desire, and attempts were made to achieve it partially by developing suitable hardware and algorithms along with specific programming such as parallelized processing according to daily needs. The hardware, which is recently used practically due to technological developments, is Field-Programmable Gate Arrays (FPGAs). Nowadays, hardware can be provided with the help of one or more FPGAs accompanied by accessories that can perform algorithms with higher calculation load. In order to implement these algorithms, it requires transferring them in a parallelized way and then designing several processors by assembling the internal structure of FPGA and finally solving the problem in a parallel manner. One of these problems is the immediate processing of image and video signals, which includes the implementation of the algorithms of image processing. Hardware design techniques, such as parallelism and pipeline, can be developed on FPGA, which is not possible with a specific design of DSP (Lee & Sobelman, 2003; Wasfy & Zheng, 2012).

During past years, many researchers implemented different hand gestures on images and a few of them implemented the gestures on hardware platforms such as FPGA. The implementation carried out in this paper differs from other one's implementation MLP neural network on FPGA using hybrid architecture. Because neural networks should always wait to know the output value from the previous layer and this value moves in front to reach the exit. But in the suggested parallel architecture, all the layers can work in parallel form with a multiplexer in the control layer section, and the speed increases very much in this state. If we deal with other methods such as the direct implementation of a digital neural network on FPGA (Dinu & Cirstea, 2007; Dinu, Cirstea, & Cirstea, 2009) and compare it with the proposed method. We will notice that direct implementation consists of three steps: a digital mathematical model of the neural network, transforming the digital model into the level of the gate, and implementation of the gates which would be time-consuming with complex calculations and causes many resources to be used.

But in the used method, parallelizing is performed in link level in the calculation unit between input - middle and middle – output. The sigmoid function is implemented in the form of hardware and a new method is suggested to keep the weights in the calculation unit, which can result in easy implementation of the neural network on the hardware. If more links can be calculated at the same time and the operation be performed on them, then the processing speed will be increased and more resources will be used. Furthermore, using shift and summation instead of multiplication (Govekar & Amonkar, 2017) and making binary the value of activation function (sigmoid function) are also among the other methods used to reduce resources and the volume of calculations. The proposed method also utilizes a layer control for multiplexing the layers to save resources.

2. RELATED WORK

As mentioned before, in recent years, different implementations with specific objectives have been carried out generally from the neural network on FPGA. In this section, we have a brief look at the parallelized and optimized implementation that resulted in reduced usage of resources. In Domen and Simon (2012), a solution was suggested to implement high-volume networks with lots of layers. Multiple makers implement each neuron and nonlinear operations, such as activating functions of the network, which are nonlinear; are turned into linear blocks, and then blocks are implemented. In this process, if, for example, an FPGA has 3600 multiple makers, it can implement a neuron of 40 $\times$ 40. Internal memories can be used to store the results of each layer along with the input and the weights. The researchers used external memories for FPGAs with limited memories.

In Lin, et al. (2010), first, two algorithms were combined to achieve optimum process, save resources and increase the speed, and then the combination was used to implement a multilayer perceptron network on FPGA. Those algorithms are as following: using multiplication of the layers and parallelizing architecture. According to the study, the layers continue to be processed in a parallel manner as opposed to the conventional state and it is not required that the results of the previous layer must be present and then sent to the next layer, rather, all the layers are involved simultaneously. When the layer *n* is calculated, the layers *n-1* and *n-2* are also calculated. The next method was used, however, is multiplexing the layers; in that, in each moment and one clock, only one layer is used according to the address of multiplex. The results of this implementation showed that the suggested method with multiplexing the layers of a neural network can reduce consumption of resources up to 30%.

In Murugan, Lakshmi, Sundar, and MathiVathani (2014), the implementation was carried out about xor problem on Virtex chip. According to the researchers, the discussed method is based on the multilayer perceptron network and using of the algorithm after diffusion, which can be used in immediate platforms such as pattern recognition, image processing, sound processing, and so forth. It consists of three main control units: after diffusion unit, pioneer unit, and the general unit, which controls two previous units. The implemented network has three layers, the first one of them, as the gate of xor, has two entrances, the hidden layer consisting of two neurons and the exit layer that has one neuron.

In Tiwari and Khare (2015), a pioneering network was implemented on the hardware platform of FPGA, Virtex series 5 using two activating functions. The innovation of the paper involving implementation is activating function using the algorithm of Coordinate Rotation Digital Computer. This algorithm becomes converge to answer by using epochs. Two activating functions were implemented by using this algorithm: sigmoid function and hyperbolic tangent function. Neurons and the relations between the layers were implemented directly by using multiple makers. The hardware language was VHSIC Hardware Description Language (VHDL), the implementation environment was Xilinx ISE and the simulator was ISim. According to the paper, the proposed method can increase the accuracy of the output results and even increases the speed of the process compared with other implementation methods.

In Amani (2013), an implementation method with a deductive assessment of resource and output speed was developed to solve the problem of working with decimal marked numbers. The percentage of saving resources and improving speed for one neuron was reported with LUT. Moreover, Amani (2013) tries to find a general formula for a neuron with several inputs so that makes it easy to estimate approximately the required resources and access speed for a multilayer neural network. This makes it possible for the designer to indicate the capacity of FPGA for a specific application. Using the suggested method involving the implementation of a neural network based on application, an example of the modulator of the spatial vector was developed for controlled initialization with the vector.

3. IMPLEMENTATION OF DIFFERENT HAND GESTURES IN MATLAB

The neural network used in this work is a four-layer MLP network with a head and two classes, one of which is for the gestures of hand and the other for the gestures of the face. Its order is 27-8-8-2, where 27 is the number of neurons of the input layer, the two hidden layers each with 8 neurons and 2 refers to two output layers. The reason for choosing 8 neurons for each hidden layer is that it results in easier implementation and less usage of resources compared with the cases with a higher number of neurons. The mentioned method in Heidaryan and Farokhi (2015) was used in the network in the form of some describing features such as Hu invariant moments and Fourier transform. Binary images consist of different hand and face gestures that are used to learn the network. The reason that every gesture of hand was not assigned to one class is due to the reduction of the parameters such as specificity, sensitivity, and accuracy in the network depending on different classes (i.e. 5 classes for 5 different hand gestures). Several networks for different gestures according to the features (single and two by two) were tested. The number of neurons for the hidden layers in the network was 4 layers as 20-8-8-2 involving the feature of Discrete Fourier Transform (DFT), 7-8-8-2 involving the feature of Hu invariant moments (Hu), and finally, 27-8-8-2 involving the feature of DFT and Hu.

3.1. Invariant moments

Invariant moments are the most important descriptors of the region and are used in many studies on pattern recognition. In recent years, this descriptor has been used widely to recognize different hand gestures. These moments have properties including displacement-invariant, rotation, scale variation, and even noise (Funatsu & Sasaki, 2018; Li et al., 2017). Of course, invariant moments are not powerful descriptors by themselves; however, they can produce a powerful neural network in pattern recognition due to the addition of other features. Apart from recognizing the gesture and shape of the images, invariant moments can be used to recognition motions. The results show that using invariant moments accompanied with other features leads to increased accuracy in the network (Premaratne, Ajaz, & Premaratne, 2013). Rank moments (p+q) and central moments are defined as follows:

$$m_{pq} = \sum_x \sum_y \; x^p . y^q . f(x,y) \tag{1}$$

$$\mu_{pq} = \sum_x \sum_y \; (x-\bar{x})^p . (y-\bar{y})^q . f(x,y) \tag{2}$$

Where:

$$\bar{x} = \frac{\mathrm{m}_{10}}{\mathrm{m}_{00}} \quad , \quad \bar{y} = \frac{\mathrm{m}_{01}}{\mathrm{m}_{00}}$$

Where, in binary images, m_{00} represents the area, and m_{01} and m_{10} represent the center of mass in the image. $f\ (x,\ y)$ shows being white or black about the pixel in a binary image and its value is 0 or 1, and x and y are the coordinates of the related pixel. Normalized central moments (Equation 3) and (Equation 4) will not change with the changes that occurred in the size of the scale image.

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}{}^{\gamma}} \tag{3}$$

$$\gamma = \frac{p+q}{2} + 1 \tag{4}$$

Finally, constant moments are defined using normalized central moments. Since the moments of upper ranks are usually sensitive to noise, only the moments up to the seventh rank are calculated as image descriptors. The equation 5 to 8 are defined as follows:

$$\varphi_1 = \eta_{20} + \eta_{02} \tag{5}$$

$$\varphi_2 = (\eta_{20} - \eta_{02})^2 + 4{\eta_{11}}^2 \tag{6}$$

$$\varphi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} + \eta_{03})^2 \tag{7}$$

$$\varphi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{8}$$

3.2. 2-D Discrete Fourier transform (DFT)

Another feature used is the 2-D Discrete Fourier transform. DFT can be used to assess signals. If we consider a digital image as a 2-D signal with limited and periodic length, then we can assess the frequency content of the image by calculating the coefficients of transforming discrete Fourier and other related coefficients (Jin, Min, Ng, & Zheng, 2019). The relations equations 9 and 10 show transforming 2-D discrete Fourier in image $M{\times}N$.

$$F(k,l) = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) e^{-j2\pi\left(\frac{km}{M}+\frac{ln}{N}\right)} \tag{9}$$

$$f(m,n) = \frac{1}{\sqrt{MN}} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} F(k,l) e^{j2\pi\left(\frac{km}{M}+\frac{ln}{N}\right)} \tag{10}$$

where, $f\ (m, n)$ shows the pixel is white or black in a binary image and its value is 0 or 1, so we have: $K, m = 0,1, \dots, M-1$ and $l, n = 0,1, \dots, N-1$ where initial 20 coefficients were used for the network.

It should be mentioned that a network is learned to recognize the color of skin, and is applied to the image. Then the binary images containing hands and faces were used to train the network and finally they turned into two matrixes of test and train. The binary images (white and black) are recalled containing only hands and faces (in white) while the remainder of the images is in black. After indicating the features based on the images and dividing them into two classes of hands and faces, two matrixes of test and train are produced from these classes. In the matrix of a database, there are 27 columns, 20 of which are related to the feature of Fourier transformation and, in fact, indicate 20 coefficients of transforming discrete Fourier. The remaining seven are related to invariant moments. As a result, a total of 27 features means that the number of our inputs or neurons is 27 and the last two columns (28 and 29) are the number of classes (1 to 2 for 2 classes).

4. THE RESULTS OF IMPLEMENTATION

According to equations 11 to 13, three parameters of specificity, sensitivity, and accuracy can be calculated for each class.

$$\text{Sensitivity} = \frac{TP}{TP+FN} \tag{11}$$

$$\text{Specificity} = \frac{TN}{TN+FP} \tag{12}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{13}$$

The database matrix of the network has 27 columns of features and 600 lines as the number of samples. In this paper, it is preferred to use the 60-40 method so that exactly 60% of the samples are assigned to the training matrix and 40% to the test matrix. As it is seen, the learning rate is 0.01, which is suitable for exiting the local minimum. The processing speed, however, is reduced and in fact, the progressing steps of correcting steps became smaller. Two numbers of the middle or hidden layer along with one output and input create four layers of the network. The network with four layers in its normal position and two hidden layers in form 8-8 has 92.31 percent of total accuracy and 0.042 of total learning error. The main reason for this increase is the very train and test matrixes and normalization applied in them. In fact, when data settle between 0 and 1 and there is less variance and discreteness, the network learns accurately and the results became better. In the following, the results and tables of the network with four layers are presented. The average accuracy of every class is 92.31% in 12000 epochs, which is the desired value. The error chart of the network with four layers was presented at the beginning of Figure 1. As it is seen, the network in 12000 epochs has the average error of 2×10^{-7} (E_{av}), where E_{av} is the average error that is a very low value. Table 1 shows the specifications of the MLP network with four layers in terms of structure and applied properties. In this table, learning is sequential, namely, in each epoch; the weights are updated, as opposed to batch learning. Table 2 shows the training error of each class along with the results of network simulation and the parameters including accuracy, sensitivity, and specificity in each class. Table 2 shows that the accuracy, sensitivity, and specificity of each class can be calculated by applying relations 11, 12, and 13. TN, FP, FN, and TP were discussed in the previous section. It is worth noting that the error presented in Table 2 is the error of testing a network with four layers, which differs from the training error of the network with the value of 2×10^{-7}. This chart is shown in Figure 1. In this chart, the quadrature axis excess represents the number of epochs and the vertical axis shows the value of average error. Of two curves, one is related to the hand and the other to face.

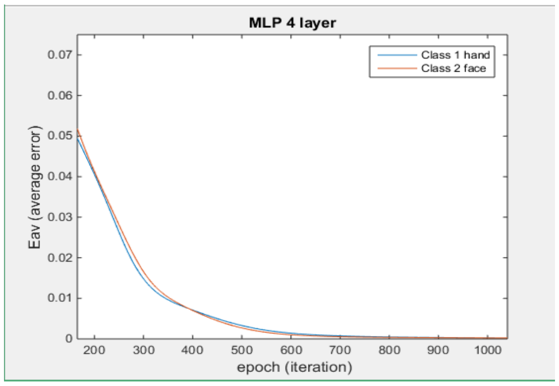


Figure 1. Chart of training error reduction for each of two classes in 1000 epochs for the network with four layers

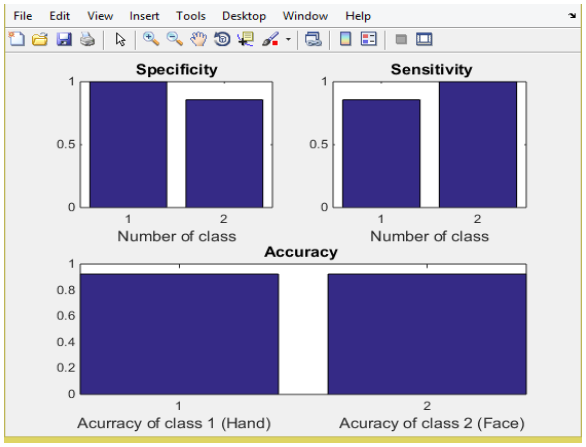


Figure 2. Bar chart of three parameters of accuracy, sensitivity, and specificity for the network with four layers.

Table 1. The properties of used four-layer MLP network

Number of neurons	20-8-8-2
Number of Features	27
Number of Epoch	12000
E_{av}	2×10^{-7}
ACC$_{av}$	92.31 %
Connection	Full connection
layers Number of	4
Learning rate	0.01
example Train	360
example Test	240
Learning way	Sequential Learning

5. FPGA IMPLEMENTATION

In this section, first, the theory of applied optimizations in the present implementation is explained and then the hardware implementation will be discussed. The proposed method for implementation of the MLP network of 27-8-8-2 include the following optimizations:

• Calculation parallelizing between the input links and hidden layer as well as neurons

• Using only binary systems of 0 and 1 as the value of the sigmoid function. In other words, one bit is used for 0 and 1 respecting the value of a function, instead of two bits.

• Using shifters instead of direct multiplication. In other words, instead of multiplying directly inputs by weights, a collector and shifter were used, which is very cheap and requires fewer resources.

• Producing open output in the form of a bit (0 and 1)

• Using a layer control for multiplexing the layers

In this case, several hidden layers are connected to a multiplexer, and the selected layer can be chosen and used at a higher speed. Figures 4 and 5 show how multiplexers of the layers are used and figure 6 shows how the layer control is connected to the neurons of middle layers as well as input and output of the block of layer control. In this case, only one layer with its neurons is implemented and the values of the neurons are displaced by a multiplexer. For example, assume that we have just one layer with 8 neurons, according to Figure 3 extracted from the reference. First, the inputs of the first layer weights are calculated by control block timing and the results are stored in the neuron named *s0* Then the value of the neuron is multiplied by the related weights by control block and multiplex and stored in the neuron named s_1. The process is continued until all the layers are calculated. When working on one layer finishes the results of calculations are stored in a look-up table, which is connected to the layer control section. Then, the calculation of the next layer starts. Thus, the network 27-8-2 is implemented, instead of network 27-8-8-8-2, and in this way; the resources of FPGA can be significantly saved.

Table 2. The results of four-layer MLP network in MATLAB

	Hu & DFT
Hand class Accuracy	92.31 %
Face class Accuracy	92.31 %
Sensitivity Hand class	100 %
Sensitivity Face class	85.71 %
Specificity Hand class	85.71 %
Specificity Face class	100 %
FN_{hand}	0
FN_{face}	10
FP_{hand}	10
FP_{face}	0
TN_{hand}	60
TN_{face}	60
TP_{hand}	60
TP_{face}	60
Number of Features	27
Number of neurons	27-8-8-2
Number of epoch	12000
E_{av}	2×10^{-7}

5.1. The structure of the calculation unit

As mentioned before, in the proposed implementation, there is only one hidden layer that can be used by a multiplexer, whenever it is needed. For example, any of the following 3 networks can be implemented using this method; a network with a layer structure of 27-8-2, a network with a layer structure of 27-8-8-2, and a network with a layer structure of 27-8-8-8-2. It continues like that only by one hidden layer with 8 neurons and one multiplexer one can implement other layers with any number of them. This is done in the control unit; thus, it will be enough to develop only the relations between the layers of input-hidden-output in the parallelized way. Figure 3 shows the structure of the network with perceptron multilayer using a multiplexer.

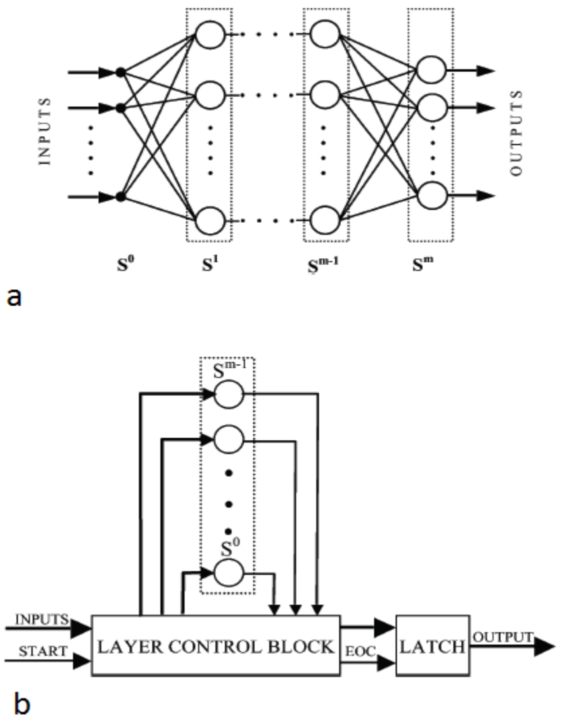


Figure 3. a) Structure of a conventional MLP network, b) Structure of MLP network with multiplexing of hidden layers.

The links between two layers can be processed independently and in a parallelized way. Data between two layers are transferred in series. It means that the next layer can have the data only when the previous layer finishes its processing on them and delivers to the next layer. Considering this, not only the links between input and middle layers but also output neurons take part in the parallelizing process. Thus, there would be two kinds of calculation:

1- Parallelizing at neuron level: this kind of parallelizing means that, for example, all the output neurons have access to the results once the neuron of the middle layer numbered one prepares them.

2- Parallelizing in link level: this kind of parallelizing means that, for example, the calculations of all the links between input neurons and the neuron of the middle layer numbered one will be done independently. No multiplexer is needed to link between bias and the neuron of the middle layer due to the bias logic was considered. The neuron in the middle layer can gain given bias using LUT. This model was shown in figure 4. In the figure, the input layer with 27 neurons, the hidden layer in the multiplex form with 8 neurons, and the output layer with two neurons were represented. Two discussed models of calculation along with multiplexing of the middle layer were also presented in the figure. The links between input and middle layers as well as output neurons take apart in the parallelizing process and the results of calculations of the middle layer are returned to the units by the diffusion. Thus, all the conditions of calculation can be assessed at the same time for the output neurons.

5.2. Hardware implementation of stimulating sigmoid function

Hardware implementation of stimulating sigmoid function is an important part of the hardware implementation of neural networks. Since the function includes nonlinear mapping and the operators such as power, sum, and division, it will take greater time and hardware resources if the implementation is applied directly.

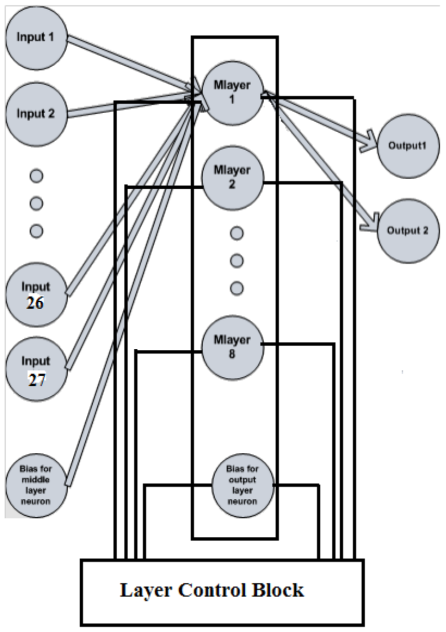


Figure 4. Structure of network implementation of 27-8-8-2 with parallelizing link and neuron and multiplex of the hidden layer.

Thus, in this method, LUT was used to implement it. In this method, what should be considered in terms of hardware implementation, is the value of the ROM unit and the complexity degree of time controlling dealing with address giving by LUT. This method, which was developed by Srdjan Coric, was showed by Equation 14 and 15:

$$a_{i(net)} = \frac{1}{1+e^{-net}} \tag{14}$$

The equation of 14 can be changed as follows:

$$a_{i(net)} = \frac{255}{1+e^{\frac{-net}{8}}} \tag{15}$$

Where net is a positive integral value. If a net is a negative integral value, the sigmoid integral value can be applied by the following formula:

$$F(net) = 255 + F(net)^* \tag{16}$$

Where $net * = 2^N + net$ and N is the input length. In the Equation of 16, F (net)* can be calculated from relation Equation of 15. The above terms show that although this method reduces the involved ROM units, it is not flexible enough. Timing of address producing becomes complex and the cycle of processing of calculation unit takes longer time due to multiple parallelizing. Investigating equation 15 shows that if the input of the sigmoid function exceeds a particular amount, its output should be nearly 255. On the other hand, if the input of the function is less than a particular amount, the output should be near zero. Thus, the following steps are suggested to increase the efficiency of the hardware implementation of the sigmoid function:

1- If the input of the function exceeds 127, its output should be considered 255 and similarly, if the input of the function is less than -128, its output should be considered zero.

2- If the input is between -128 to 127, the output can be calculated by the following relation:

$$a_{i(net)} = \frac{255}{1 + e^{\frac{-net}{24}}}$$

These values are stored in LUT. While working with the neural network if the number 128 is added to the input, it can produce easily the given address in LUT. The investigations show that the suggested method is suitable for middle and output layers. Furthermore, networks that use this implementation method are performance as much as the networks, which calculate the sigmoid function directly with float calculating data.

5.3. Implementation of links

Four RAMs were used to implement links, multiple the weights in the input, and bias the neurons: a RAM related to the weights of links between input and middle layers, a RAM related to the values of middle layer bias, a RAM related to the weights corresponding to the links between the neurons of middle and output layers, and a RAM related to the values of output neurons bias. The number of neurons in the middle layer is showed by N. Thus, RAM is placed in the calculation model of link in a way that the values of weights of the links between the neurons and the number one neuron of the middle layer can be placed in the zero address of the RAM. The other weights are placed in the same manner. Totally, N address is required. After calculating the weight of links and the bias values on the learning phase by MATLAB software, the results are placed in FPGA by using the rules mentioned above. All links of the neural network are accessible by a sequential increase of the address. This was shown in figure 5. When the address is zero, the operations shown on the left side of the figure should be performed. All links presented by the arrow on the left side of the figure are to be covered. With increasing the address up to 1, more links can be covered. These links were presented by the arrow on the right side. Thus, all the links of the network can be accessible by increasing the address from zero up to N (N is the number of neurons in the middle layer).

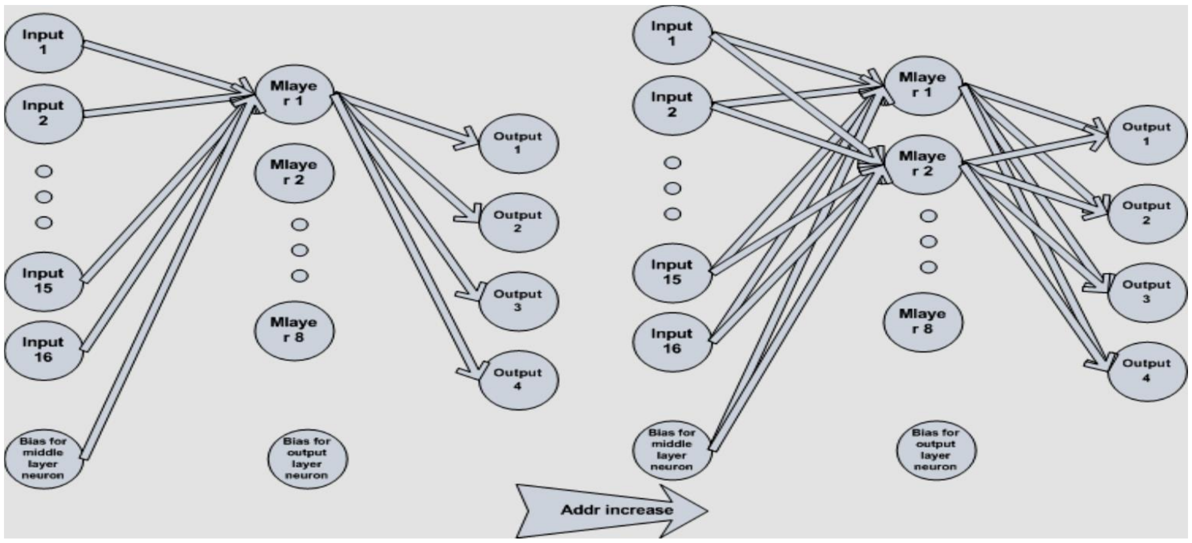


Figure 5. evaluating network links regarding memory address increase

5.4. Implementation Modules

Verilog language was used to implement the related modules. Each neuron has one module in each layer. As an example, each neuron of the hidden and output layers has a separate module for itself. As discussed before, for multiplication, the operations of shift and sum are used instead. This algorithm is used while multiplying booth about which we will discuss in the next section. Four RAMs were used to maintain the values of the weights and bias of neurons and in continue, we will discuss that. Finally, one module named control unit, undertakes the task of controlling all of the modules and multiplexing the middle layers.

5.4.1. Booth multiplication

Given the optimizations presented in section 3, shift and sum existed in the algorithm of booth multiplication, were used to multiply the weights in the inputs. The algorithm of booth multiplication introduces a procedure for multiplying binary numbers in presenting the compliment with 2 marks. The basis of the algorithm is that the strings of 0 in the multiplier have not to be summed, rather they need only to be shifted displaced. The strings of 1 in multiplier, from bit order of 2^k to 2^m, can be considered as $2^{k+1} – 2^m$. Booth algorithm, like as all methods of multiplication, has to consider multiplier bits and shift the product partially. Before shifting, the multiplier may be summed with or reduced from the partial product or it may remain unchanged. Based on the discussed matters, to do calculating operations in each neuron we need the implementation of multiplication. In this respect, the algorithm of booth multiplication was used. Both inputs and weights are considered 4-bit and the result 16-bit.

5.4.2. The neuron of the middle layer

In the module of the middle layer, the inputs are considered 4 bits and since there are 27 inputs, we have a total of 108 bits from zero to 107 all of which are assigned to the inputs and their corresponding weights. The module of the neuron in the middle layer can be assumed as in figure 6. HZ_bus, start, clk are the single-bit input signals for the starting clock of calculating operations, neurons of impedance (HZ), the lines of exit bus belonged to the neuron, respectively. Weight_Mid is the 108 bits input from one line address of RAM related to the weights of the links between the input and middle layers. These 108 bits are the length of Weight_Mid. Input_Mid is the 108-bit input consisting of 27 input values any of which is 4-bit. In the network implementation section of MATLAB software, it should be notice that the input value cannot be higher than 16. Bias-Mid is the 16-bit input coming from the RAM relating to the bias of the neurons in the middle layer and having the bias value corresponding to the related neuron. Output_Mid is the 8-bit output of the neuron, which is given to the next layer and is considered from zero to 7. The reason for it being 8 bits is that two 4-bit numbers are multiplied by each other and the result will be 8-bit.

5.4.3. The neuron of the output layer

The module of the neuron in the output layer can be imagined as Figure 7. The description of this module is the same as that of the neuron in the middle layer with a difference. In this case, the values of the weight of links and the bias of neurons result from the related RAMs. Moreover, Input_Out is an 8-bit input that, in fact, is the very value of the 8-bit output of the neuron of the previous layer and relates to the module of the middle layer.

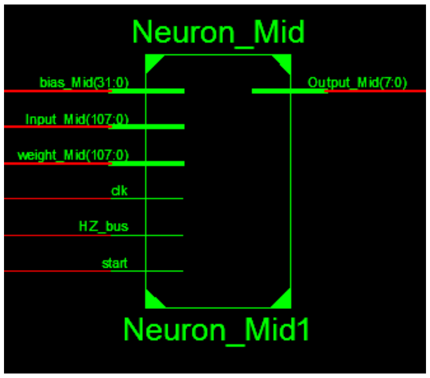


Figure 6. The module of the neuron in the middle layer

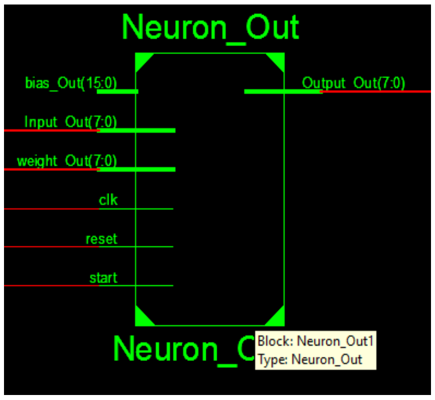


Figure 7. The module of the output neuron.

5.4.4. RAM of the weights corresponding to the links between the input and middle layers

The module of this RAM can be considered as figure 8. In this module, the input signal is used to give initialization values to the memory. These values are placed in the memory as the network was trained in MATLAB software. The 3-bit input of Add indicates the given address. 4 bits are assigned to each value and there are 27weights (the number of inputs) and at the end, the 108-bit data of output contains the weights. (It is also possible to use a doubled space with 256 bits for more neurons of the middle layer) the module of next RAMs can be implemented similarly including the RAM related to the weights corresponding to the links between middle and output layers, the RAM related to the bias corresponding to the neurons of the output layer, and the RAM related to the bias corresponding to the neurons of the output layer. The only difference is that their output data is different and based on applying so called RAM that we avoid referring to it.

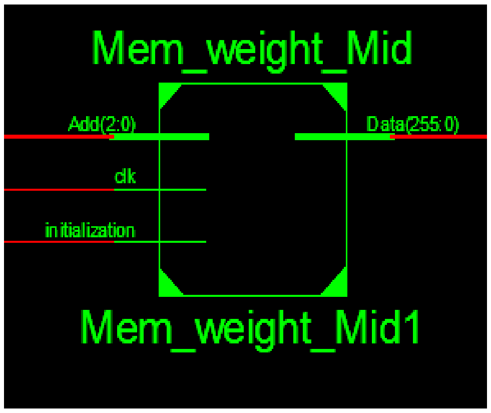


Figure 8. The RAM related to the weights corresponding to the links between middle and input layers

5.4.5. Control unit of neurons and layers

In fact, the control unit of neurons and layers has two important tasks to do. 1- Controlling neurons and layers and allowing the modules to start and stop on the condition of reaching inputs to outputs. 2- Controlling layers through multiplexing them. The module of the control unit can be considered as figure 9. This unit controls the start of calculating operations of the neurons in the middle layer by 8 output signals start_Mid$_1$ to start_Mid$_8$. The output signal (initialization) activates the corresponding signal located in the memories to give the initial values. 8 output signals of HZ_bus$_1$ to HZ_bus$_8$ have the responsibility of managing the order of being impedance of output bus by the neurons of the middle layers. The address lines of Address_Mid and Address_Out are to produce the proper address of the memories containing the corresponding weights to the links. The signal of end_work indicates the end of the network task. This unit receives an input named layer_number to control the layers. In fact, the input indicates the number of hidden layers, which is changeable from 1 up to 3. The operations of control unit should be repeated based on the number of the hidden layers, that is, if we have two hidden layers, to use multiplexing the layers, the operations of the control unit should exactly be repeated twice, in that, we have a layer with two timings, instead of two hardware layers. In figure 9, the number of layers was applied as an input in the module.

5.4.6. The module of implementation neural network of 27-8-8-2

This network is implemented by the modules introduced in the previous section. What is noticeable about this module is that a look-up table and multiplexer module were used. In fact, the task of a multiplexer is to choose the output of the neurons in the middle layer in the look-up table. The output of the neurons in the middle layer is stored in the look-up table. The multiplexer chooses the output of the neurons in the middle layer depending on the number of hidden layers. As an example, if we have two hidden layers, the results of the neurons in the middle layer are stored twice and the multiplexer chooses the results of the neurons in the middle layer and then gives them as output to the neurons of the output layer. In continue, the schematic of both networks was shown. Figure 10 shows the schematic of the 27-8-8-2 network along with wiring and relation among the internal components.

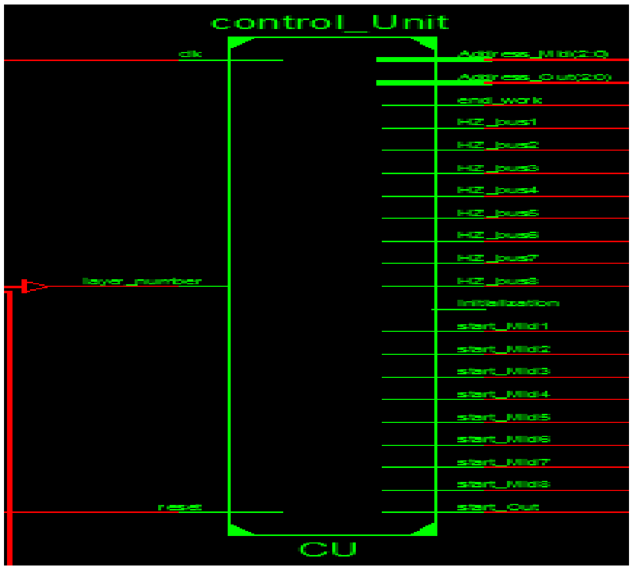


Figure 9. The module of control unit.

6. SYNTHESIS AND SIMULATION

Two different types of hardware were used in this implementation. The differences can be compared in the references. 1-Spartan 3 XC3S5000 5fg900 with limited resources. 2- Spartan 6 slx 100t with larger resources. The synthesis was done by Xilinx Synthesis Tool (XST), existed in ISE software package, and is used to both designing processes based on HDL or schematic. In this section, we discuss the simulation of the 27-8-8-2 network after synthesizing. Before starting, it should be noted that simulation is done in two steps. First after the synthesis to evaluate the accuracy of circuit function and second after place and route phase to check the information of delay in the blocks along with the routes after routing and evaluating accurately the behavior of the circuit under the worst conditions, and finally evaluating the arrangement of the project after laying out and route location. simulation method that is discussed in this section involves before locating and routing steps. From this section, all simulations are done by ISE simulation tool of software package, namely Isim.

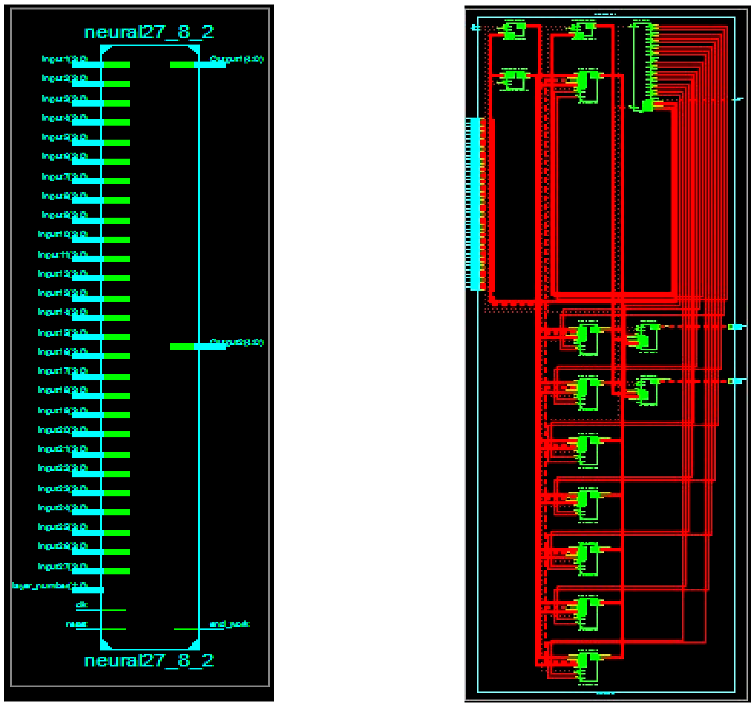
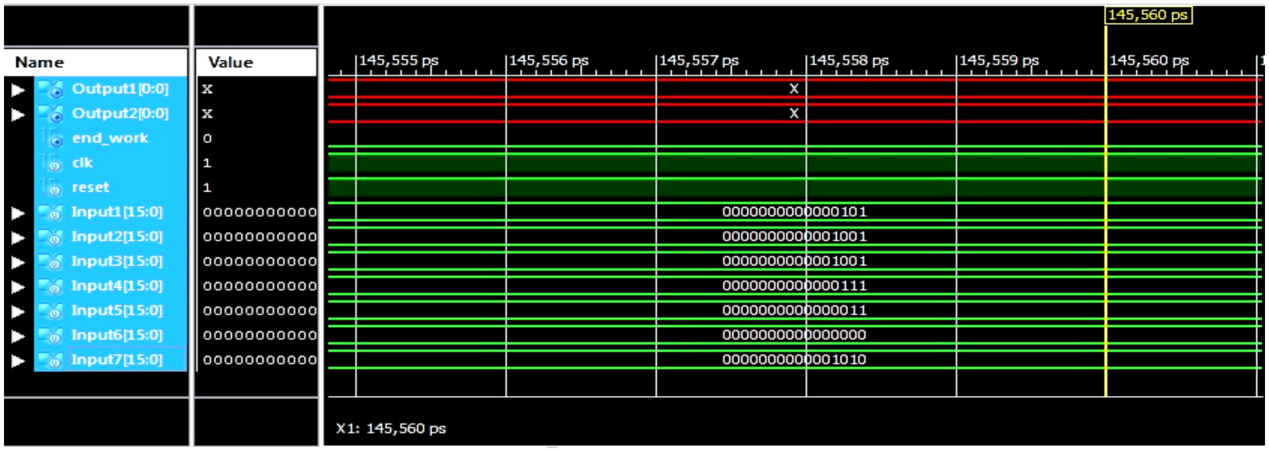


Figure10. Schematic of 27-8-8-2 network, wiring, and relation among its components

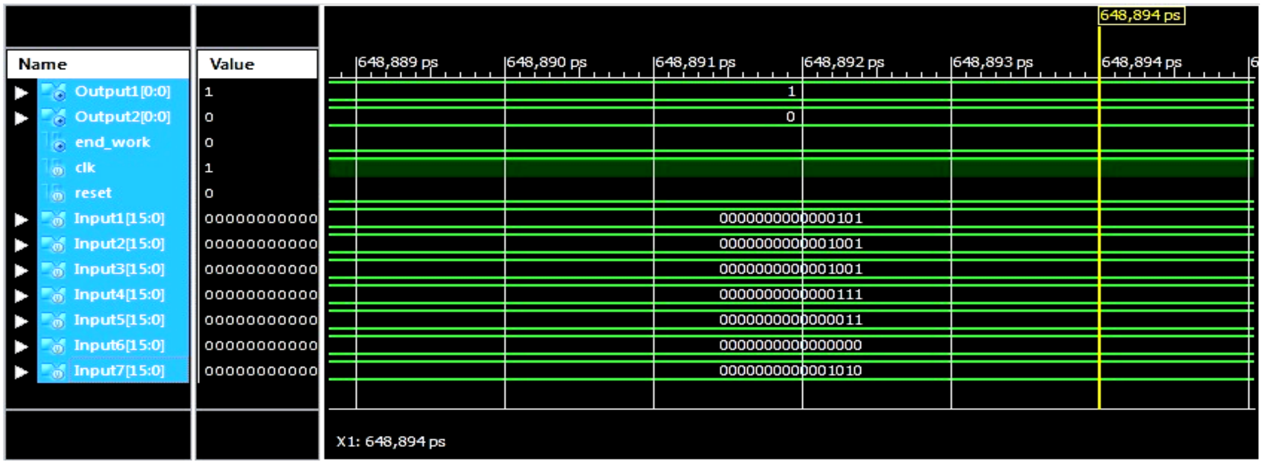
We need a test file to do the simulation. The test file consists of simulation information including the values of inputs and circuit variables, total time of simulation, information of Clock, and so on. In this implementation method, the implementation module of the sigmoid function uses only the Unisim library and since Isim simulator compiles this library during application automatically (version higher than 10), there is no need to compile it manually. 27 inputs have the values and one of the lines of values is the test matrix in MATLAB software. The result of the network according to the values is in the form of 0 and 1 meaning that Output 1 is 1 and Output 2 is 0. The Reset signal has the value 1, at first; however, the outputs remain unknown and become zero in 150 Nano seconds. Once they become zero the network starts working. The circuit clock changes its position every 5 Nano seconds. The network needs 650 Nano seconds altogether to reach its final value and after that, the outputs become stable. Figure 11 (a and b) shows the results of Isim simulator (in this simulation, the inputs have 16 bits).

7. EVALUATION OF IMPLEMENTATION IN 27-8-8-2 NETWORK

This section provides a summary of reports about mapping, route, and implementation on Spartan 3 XC3S5000 5fg900, in the 27-8-8-2 network. First, a neural network was implemented without multiplexing the layers and by using directly multiplication instead of the booth multiplication algorithm The results of this process were presented in table 3. Then a 4-layer neural network was implemented with multiplexing the layers and the optimizations discussed in the previous section. The results were compared with those of the previous network in table 3. As seen in Table 3, using multiplexing of the layers prevents resource waste significantly. In the last column, for all the resources, the lookup table involving the hardware platform was optimized 24%. Considering this matter, complicated functions such as hyperbolic tangent can be used in activating function, which brings about better results.



```
Name            Value
Output1[0:0]    x
Output2[0:0]    x
end_work        0
clk             1
reset           1
Input1[15:0]    00000000000
Input2[15:0]    00000000000
Input3[15:0]    00000000000
Input4[15:0]    00000000000
Input5[15:0]    00000000000
Input6[15:0]    00000000000
Input7[15:0]    00000000000
```



```
Name            Value
Output1[0:0]    1
Output2[0:0]    0
end_work        0
clk             1
reset           0
Input1[15:0]    00000000000
Input2[15:0]    00000000000
Input3[15:0]    00000000000
Input4[15:0]    00000000000
Input5[15:0]    00000000000
Input6[15:0]    00000000000
Input7[15:0]    00000000000
```

Figure 11. a) Simulation results of 27-8-8-2 network (network of recognizing hand) in time that Reset signal is 1. (Red color indicates unknowingness of the outputs). b) In final time (the whole neurons are calculated and the output become stable).

Module Name	Optimized Neural 27-8-8-2 (The proposed work)			Ordinary Neural 27-8-8-2 (The previous work)			Optimized percent
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization	

Table 3. Comparing the mapping results and route location in 27-8-8-2 network (network of hand detector) on Spartan 3 XC3S5000 5fg900 by using directly multiplexing the layers and optimization.

Module Name	Optimized Neural 27-8-8-2 (The proposed work)			Ordinary Neural 27-8-8-2 (The previous work)			Optimized percent
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization	
Number of slice Flip Flop	8260	66560	12%	12762	66560	19%	7%
Number of 4 Input LUTs	23296	66560	35%	38144	66560	57%	22%
Number of occupied Slices	14165	33280	42%	22279	33280	66%	24%
Total Number of 4 Input LUTs	26103	66560	39%	41940	66560	63%	24%
Number of bonded IOBs	121	633	19%	357	633	56%	37%
Target Device	Spartan3 XC3S5000 5fg900			Spartan3 XC3S5000 5fg900			

8. CONCLUSIONES

In this paper, a method was developed to recognize hand on FPGA by using a neural network with a high number of hidden layers in the form of a multiplex. The results show selecting the number of the neurons and layers is important. In the optimizations applied on the suggested method, 71945 resources were used in the FPGA platform which resulted in saving resource consumption about 37.7 %, compared to the direct implementation (115482 resources).

REFERENCES

Amani, B. (2013). *Neural Network implementation using FPGA.* Paper presented at the 5thIranian Conference on Electrical and Electronic Engineering.

Dinu, A., & Cirstea, M. (2007). *A digital neural network FPGA direct hardware implementation algorithm.* Paper presented at the 2007 IEEE International Symposium on Industrial Electronics.

Dinu, A., Cirstea, M. N., & Cirstea, S. E. (2009). Direct neural-network hardware-implementation algorithm. *IEEE Transactions on Industrial Electronics, 57*(5), 1845-1848.

Domen, V., & Simon, B. (2012). Implementation of Massive Artificial Neural Networks with Field-programmable Gate Arrays. *IFAC Proceedings Volumes, 45*(4), 133-138.

Funatsu, T., & Sasaki, N. (2018). *Study of Measurement Method in Inter-Vehicle Distance Using Hu Moment Invariants.* Paper presented at the 2018 18th International Conference on Control, Automation and Systems (ICCAS).

Govekar, D., & Amonkar, A. (2017). *Design and implementation of high speed modified booth multiplier using hybrid adder.* Paper presented at the 2017 International Conference on Computing Methodologies and Communication (ICCMC).

Heidaryan, M., & Farokhi, F. (2015). *Robust hand gestures tracking method in cluttered background based on multilayer perceptron.* Paper presented at the 2015 5th International Conference on Computer and Knowledge Engineering (ICCKE).

Jin, Z., Min, L., Ng, M. K., & Zheng, M. (2019). Image colorization by fusion of color transfers based on DFT and variance features. *Computers & Mathematics with Applications*, *77*(9), 2553-2567.

Kirishima, T., Sato, K., & Chihara, K. (2005). Real-time gesture recognition by learning and selective control of visual interest points. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(3), 351-364.

Lee, H., & Sobelman, G. E. (2003). Performance evaluation and optimal design for FPGA-based digit-serial DSP functions. *Computers & Electrical Engineering, 29*(2), 357-377.

Li, X., Wang, X., Xie, D., Wang, X., Yang, A., & Rong, M. (2017). Time–frequency analysis of PD-induced UHF signal in GIS and feature extraction using invariant moments. *IET Science, Measurement & Technology, 12*(2), 169-175.

Lin, Z., Dong, Y., Li, Y., & Watanabe, T. (2010). *A hybrid architecture for efficient FPGA-based implementation of multilayer neural network.* Paper presented at the 2010 IEEE Asia pacific conference on circuits and systems.

Murugan, S., Lakshmi, K. P., Sundar, J., & MathiVathani, K. (2014). Design and Implementation of Multilayer Perceptron with On-chip Learning in Virtex-E. *AASRI Procedia, 6,* 82-88.

Premaratne, P., Ajaz, S., & Premaratne, M. (2013). Hand gesture tracking and recognition system using Lucas–Kanade algorithms for control of consumer electronics. *Neurocomputing, 116,* 242-249.

Tiwari, V., & Khare, N. (2015). Hardware implementation of neural network with Sigmoidal activation functions using CORDIC. *Microprocessors and Microsystems, 39*(6), 373-381.

Wasfy, W., & Zheng, H. (2012). General structure design for fast image processing algorithms based upon FPGA DSP slice. *Physics Procedia, 33,* 690-697.