Journal of Engineering Sciences
Assiut University
Faculty of Engineering
Vol. 46
No. 5
September 2018
PP. 539 – 555

EFFICIENT HARDWARE IMPLEMENATATIONS OF FEEDFORWARD NEURAL NETWORKS USING FIELD PROGRAMMABLE GATE ARRAY

Mohamed H. Essai [1, *], Marina Magdy [2]

[1] *Electrical Eng. Depart, Faculty of Engineering-Qena, Al-Azhar University, Egypt.*
[2] *Electronics & Communications Engineering Department,*
Higher Institute of Engineering and Technology, Luxor – El Tod, Egypt.

Received 30 April 2018; Accepted 21 May 2018

ABSTRACT

Hardware implementation of Artificial Neural Network (ANNs) depends mainly on the efficient implementation of the activation functions. Field Programmable Gate Array is the most appropriate tool for hardware implementation of ANNs. In this paper we introduce FPGA-based hardware implementation of ANNs using five different activation functions. These implemented NNs are described using Very High Speed Integrated Circuits Hardware Description Language (VHDL) and carried out by Digilent Basys 2 Spartan-3E FPGA platform from Xilinx. The performances of the implemented NNs were investigated in terms of area efficient implementation, and correct prediction percentages for solving XOR, and Full-Adder problems.

Keywords: Artificial system, VHDL, FFNN, FPGA, Back-propagation, Activation function.

1. Introduction

Neural networks (NNs) are widely used in many areas, both for systems development and applications. NNs are frequently used to solve numerous problems that cannot be handled by other hard computing methods. The artificial neural network (ANN) has been used in many applications in the field of science and technology [1]. The ANN capabilities such as mapping, modelling and classifying the nonlinear systems encouraged the researchers to construct artificial intelligent (AI) systems that utilize the current progress in high speed computing. If the practical implementation of high speed and cheap artificial neural computation systems are possible; many real time applications will be resolved. To achieve this goal, numerous proposals have been presented on the implementation of ANNs [2].

ANNs implementations can be categorized into: Software (SW) implementations and Hardware (HW) implementations [3]. The SW implementations offer flexibility (easy to implement), and reliability but provide poor performance. The implemented ANNs that use

* Corresponding author.
E- mail address: *mhessai@azhar.edu.eg*

SW approach are simulated, trained, verified, and tested using the general purpose sequential (Von-Neuman) computers for modelling a variety of NNs.

The HW implementations are generally difficult and consuming more time to be built but provide better performance in comparison with SW versions [4].

There are analog, digital and hybrid system architectures offered for the HW implementations of ANNs. Analog architectures are more accurate, but difficult to be implemented and have some of problems with weight storage. Digital architectures have the advantages of high flexibility, high accuracy, better replicability, low noise sensitivity, better testability, and weight storage does not become a problem.

The digital HW implementations of ANNs can be classified as (i) Field Programmable Gate Array (FPGA)-based HW implementations (ii) Digital Signal Processors (DSP)-based HW implementations (iii) Application Specific Integrated Circuits (ASIC)-based HW implementations. A DSP based HW implementation is sequential and therefore does not maintain the parallel architecture of the ANs in a specific layer. ASIC- based HW implementation does not support user re-configurability [5].

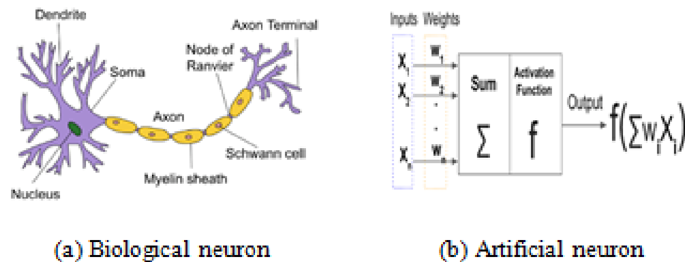
FPGA is a suitable HW tool for implementing ANNs, because it preserves the Parallelism characteristic of the neurons in the layer and provides flexible reconfiguration. Parallelism, modularity and quick dynamic adaptation of both ANNs weights and topologies are three computational characteristics, usually related to ANNs.

In this paper five FPGA-based Feedforward Neural Networks (FFNNs) have been described (coded) using VHDL, and implemented using Xilinx Spartan-3E FPGA platform, for solving XOR problem (as a case study) with less implementation area and acceptable prediction error percentages. The learning process follows the chip-in-the-loop learning mechanism which entails both the implemented ANN for carrying out the computations and Back Propagation (BP) learning algorithm for updating the ANN's weights.

The rest of the paper is arranged as follows: Section 2 reviews the basics of artificial neuron networks, challenges of ANN hardware implementations, and ANN hardware realizations techniques. Section 3 introduces the most common learning mechanisms. Section 4 summaries the hardware implementation Issues. Section 5 explains what activation function is, activation functions' characteristic, and implemented activation functions. Section 6 explores Implementations' conditions and results. We conclude in section 7. Finally future work is introduced in section 8.

2. Artificial neural networks

ANNs offer the mathematical representation of the biological neural system. A biological neuron in Fig. 1.a, receives incoming signals through its dendrites, and passes these signals to the body of cell. Axon transmits the signals to the synapses, which are the bonds of the axon of the cell with the dendrites of another cell [6]. The artificial neuron is shown in Fig. 1.b, can be considered as a data processing unit in three stages; weighing its input values, summing them all and filtering them using the activation function.



(a) Biological neuron (b) Artificial neuron

Fig. 1. Structural diagrams of biological neuron and artificial neuron.

ANNs can be divided into two main categories: Feedforward Networks (FFNNs), and Recurrent Networks (RNNs). In FFNNs the computations are executed in a layer-by-layer fashion as shown in Figure 2. RNNs have an interconnected network structure including cycles (feedback loops) as shown in Figure 3. This permits it to show dynamic temporal behavior. Unlike FFNN, RNNs use its internal memory to process arbitrary input sequences [7]. This enhances the applicability of RNNs to numerous tasks such as non-segmented, associated handwriting recognition or voice recognition.

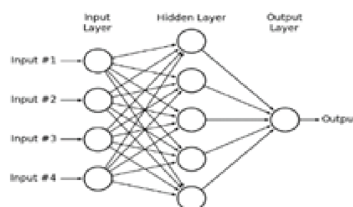


Fig. 2. Feedforward network structure.

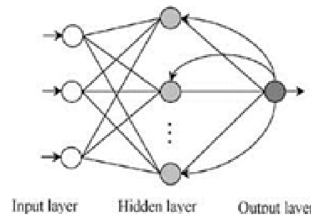


Fig. 3. Recurrent network structure.

ANNs can be trained using one of two training approaches; supervised training algorithm or unsupervised training algorithm. Supervised training algorithm requires a trainer to present both inputs and actual outputs (i.e. input/output training pattern) which aid the trainer to determine the correct relationships between the training patterns of a given problem a priori. Unsupervised training algorithm does not require the trainer to provide the expected results [8].

The Backpropagation (BP) training algorithm is a method for updating the weights of the multilayer FFNNs. The BP training algorithm simulates a given function by changing the internal weights of the input signals to obtain the accepted output signal. In the supervised learning method, the error between the estimated output of the system and the actual output is presented to the system and used to update its state [8, 9].

2.1. Challenges of ANN hardware implementations

In artificial neural systems, synapse acts as a network wiring. The number of synapses is scaled quadratically with the number of neurons, which will not be practical for wiring [10]. Today's integrated circuit manufacturing technologies are basically multi-layered 2D structures, which further limits the possibility of a complete connection. Therefore, most implementations restrict the wiring to some neighborhood of each neuron [11].

Synaptic weights must be defined accurately so as to ensure good convergence of the training algorithms. Due to the training process synaptic weights can be updated more and more time before the optimal performance is achieved, and the training process is stopped.

Neuron, where most of the mathematical operations such as summation of weighed inputs

and calculation of nonlinear activation function are performed. Activation function is the most important component in the neuron structure. There still open questions about what is the most appropriate activation function for FPGA-based ANNs? what is the robust activation function? and what is the activation function that achieves the lowest implementation area?

2.2. ANN Hardware realizations

2.2.1. General purpose parallel computers

Most modern AI systems are considered to run on the sequential (Von Neumann) computer architectures. The performance of these architectures degrades quickly as knowledge is added to these computer systems, also these architectures do not perform well in real time applications. To speed up the implementation of AI programs, several methods have been identified. Parallel computing (Parallelism) was used to speed up existing AI systems [12].

Fine-grain parallel implementations on massively parallel computers [13] can either be single-instruction stream, multiple-data stream (SIMD), or multiple-instruction stream, multiple-data stream (MIMD). Hence their drawback is the connectivity of standard neural models that influenced in costly information exchanges.

Parallel implementations of Coarse grain are mostly applied to neural training, so their effectiveness depends on the sequence of standard learning algorithms such as stochastic gradient descent [14]. In addition, massive parallel computers are expensive resources and cannot be used in embedded systems. Such solutions are not preferred for neural structures and complex neural calculations or training methods [15].

2.2.2. Analog Application Specific Integrated Circuits (A-ASIC)

Many analog HW implementations have been performed. They are very fast, dense and low-power, but they suffer from some of specific problems such as low precision, inefficient data storage, un-robust [16], and difficulties in on-chip learning [17]. ASIC is not cheap or flexible solution, and its development is tedious [18]. ASIC is very confusing for users who are not analog technology minded.

2.2.3. Digital application specific integrated circuits (D-ASIC)

Many digitally ICs have been used in the design of ANNs. D-ASICs offer higher accuracy, higher robustness, and can process efficiently all the required neural computations. A lot of efforts are needed to design working D-ASICs, and it's extremely expensive when a small number of ICs are needed (ASIC shortcomings). Usually, D-ASICs are used to implement limited parts of NNs in order to be embedded in neurocomputer systems [19]. Nevertheless, the structure of this NN can't be mapped directly onto the IC [20].

2.2.4. Field programmable gate arrays (FPGA)

FPGA is a type of programmable logic (PLD) devices, which offer design flexibility like software, but with performance rates closer to D-ASICs [21]. In addition to the ability to be reconfigured frequently after being manufactured, hardware designers have been used FPGAs as a prototyping tool. Due to enhanced performance, FPGAs have grown over the years, and therefore they are used in reconfigurable computing applications.

FPGAs are chosen for the ANNs implementation due to the following reasons:

- FPGAs contain a large number of logical gates, ranging from tens of thousands to several million the logic gates.

- FPGAs can be configured to modify the logical function when embedding in the system.
- FPGAs have a short cycle of design, which leads to a cheap logic construction.
- FPGAs have a parallel computing environment.
- FPGAs have powerful design entry (HDL, or schematic), synthesis, and simulation tools.

The ANNs architectures should be entered schematically or algorithmically at the initial stage of the FPGA-based system design. When an ANN-based FPGA system is developed, indicate the ANN architecture from a symbolic level. This level enables us to use VHDL, which is a type of the hardware description programming languages [22]. VHDL supports many levels of instructions and enables you to accurately describe electronic circuits, from the simplest combinational logic circuits such as adders and comparators to the microprocessors.

Table 1, outlines the advantages and disadvantages of the commonly used solutions for ANNs implementation. Each solution is estimated approximately for each realized aspect.

Architecturally, FPGAs can be described as an array of separated configurable logic blocks (CLBs) that can be interconnected in a general way and programmed by the user. FPGAs contain three main blocks, which are CLBs, I/O blocks and connecting blocks, these blocks are used to describe any FPGA, since these blocks vary from one FPGA vendor to another. Logical blocks carry out a logical function. The connection blocks connect the logic blocks to the I/O blocks [23].

In particular, Xilinx traditionally produces FPGA based on SRAM; so-called because of programmable resources for these types of FPGAs, are controlled by static RAM cells.

Table 1.
Suitable / unsuitable devices for implementing neural networks [21].

Device / Comparison	A-ASIC	D-ASIC	FPGA	Processor Based	Parallel computer
Speed	highly convenient	very convenient	convenient	inconvenient	convenient
Area	highly convenient	very convenient	convenient	inconvenient	very inconvenient
Cost	very inconvenient	very inconvenient	very convenient	very convenient	very inconvenient
Design time	very inconvenient	very inconvenient	very convenient	very convenient	convenient
Reliability	very inconvenient	convenient	very convenient	very convenient	very convenient

Fig. 4, shows the basic construction of the Xilinx FPGA [24]. It consists of a 2D array of CLBs. Horizontal and vertical Interconnectors (routing channels), are found between the rows and columns of CLBs. Note that the CLBs and routing architectures are different for each FPGA generation and family. In this paper Digilent Basys 2 FPGA development board was used, this board built around a Xilinx Spartan -3E FPGA and an Atmel AT90USB2 USB controller as shown in Fig. 5.

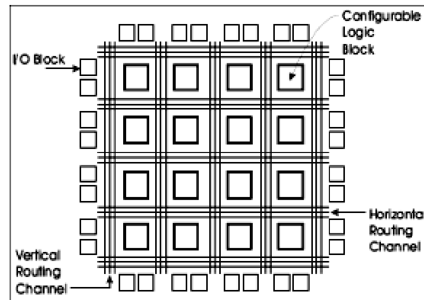


Fig. 4. General Architecture of Xilinx FPGAs.

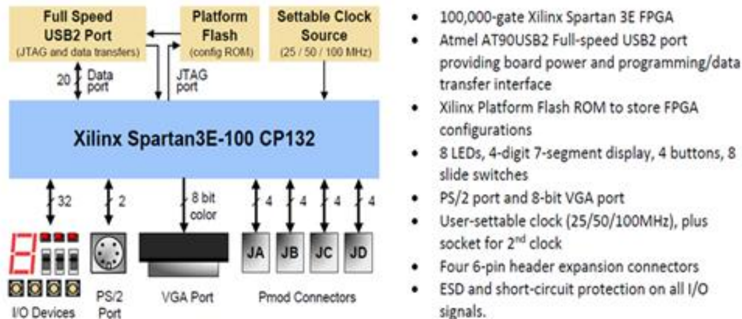


Fig. 5. The Digilent Basys 2 board block diagram and features.

3. Learning mechanisms

A common learning mechanism for most ANNs covers both the training and recall phases. In the training phase, the network weights are updated in accordance with the application. In the case of the perceptron, this involves using the backpropagation algorithm on a classified training set; in the case of associative memory this involves adjusting the weights to ensure the necessary memories act as local attractors.

At the recall phase, the new inputs are guarantee, and the network is allowed to balance (single-pass forward for the perceptron and evolution to equilibrium for associative memory, for example). Although the recall phase is always performed on the physical network itself, the training phase can be performed previously.

There are three crucial training mechanisms [10]:

1) Off-chip learning involves the execution of the learning phase using software based-simulated network. This makes it possible to make learning calculations faster and more accurate than can be carried out with the help of a HW network. However, manufacturing alteration of the HW is not considered.
2) Chip-in-the-loop learning includes both the HW network and external SW calculations. It depends on the software to execute the learning algorithm but uses the HW network to perform the calculations. In the case, for example, of the backpropagation algorithm, the forward pass is performed by the network, while weight updates are performed in the software. Therefore, the accuracy of the computation is not limited to the hardware capabilities, but the actual behavior of the network is still taken into account.
3) On-chip learning, this approach uses only a hardware chip for training. The On-chip learning is slower and less accurate in calculating the weights than the other two

methods. This method does not include external manipulations at the learning stage in advance, and this makes it more realistic for embedded equipment, and networks designed with this approach will be able to update their training throughout the entire time. However, the design is inherently more complex and less flexible, since the algorithm that performs the training must be implemented on hardware.

4. Implementation issues

FPGA-based NN implementations should deal with the problems of the digital HW implementations of NNs' applications such as, topology-related issues and area-related issues. Many efforts have been made to solve such these issues, and the related issues become more acute due to the specific limitations of FPGAs.

We focus on area saving issues that can be solved by [21]:

- An activation function optimized implementation (for example, a cord-like algorithm or piecewise polynomial approximation, can be controlled by fast multipliers or look-up tables). The choice of the optimal implementation of the activation function depends on some characteristics of the application such as the required accuracy or the required chip (in, on, or off) training.
- Bit serial arithmetic may be standard or optimized or serial arithmetic online. The best choice of which arithmetic approach is more appropriate for digital HW implementations of NNs is discussed previously in [25] but determining the type of FPGAs is not taken into account, and online arithmetic is not considered. It is well known that serial arithmetic requires more clock cycles and it is recommended to be used with the pipelined operators.
- Arithmetic based on pulses; provides tiny operators, so a medium sized NN can be completely constructed entirely on a single FPGA, but restrictions on the routing of FPGA don't allow the implementation of the large NNs.

In the context of area-related implementation issues, in this paper, we are planning to implement more efficient neural networks that achieve high precision, with less area occupation on FPGA. Thus, five feedforward neural networks have been designed using five different activation functions. The performance efficiency of the implemented FFNNs will be investigated through solving the problems of XOR, and Full-Adder. XOR gate is used frequently in building arithmetic logic circuits such as adders, computational logic comparators and error detection circuits such as parity checker. Also, Adders are important in computers and other digital systems in which numerical data are processed.

5. Activation function

The activation function is one of the most important components of the ANN. It basically determines whether the neuron should be activated or not. Regardless of whether the received information by the neuron relates to the given information or it should be ignored.

$$Y = Activation\ (\sum(weight * input) + bias) \tag{1}$$

If the activation function does not exist, the weights and bias will simply perform a linear transformation. It is easy to solve the linear equation but its ability to solve complex problems is limited. In the absence of the activation function, NN acts as a linear-regression model. The activation function performs a nonlinear conversion to the input, which allows it to learn and execute more complex tasks. It is desirable that NNs deal with

complex tasks, such as image classifications and language translations. Such tasks are never being performed by linear transformations.

Using the differentiable nonlinear activation functions make the use of the back-propagation is feasible where the gradients and errors are provided in order to update both weights and biases [26].

Typically, activation functions can be classified into two types. The first type is a linear activation function, where it is a simple linear function of the form $f(x) = x$. Basically; the input passes to the output without any variation.

The second type is the non-linear activation functions which are used to separate the data that is not linearly separable and are the most used activation functions. A non-linear equation manipulates the mapping from inputs to outputs. Sigmoid, tanh, relu, lrelu, prelu, and swish are examples of the non-linear activation functions.

Activation functions properties can be summarized as follows [27]:

- Nonlinear - Two-layer NN with non-linear activation function acts as a universal function approximator.
- Continuously differentiable – This is a preferable feature for enabling optimization methods that based on gradient.
- Range – If the activation function has a finite range, the gradient training methods tend to be more stable. While if the activation function has an infinite range, the training process becomes more effective, in this case lowering the training rates is significant.
- Monotonic – If the activation function has a monotonic behavior, the error surface of a single-layer model will be convex.

5.1. Implemented activation functions

5.1.1. Sigmoid activation function

A sigmoid function is a mathematical function having an "S" shaped curve (sigmoid curve). It is a special case of the logistic function. It is commonly used in ANN, due to its monotonous character and the ability to derivation which make them suitable for training algorithms. Its usages in neural network are:

1. Activation function that transform linear inputs to nonlinear outputs.
2. Bound output to between 0 and 1 so that it can be interpreted as a probability.
3. Make computation easier than arbitrary activation functions.

This function takes the input and squeezes the output into the range 0 to 1, according to the following expression:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The log-sigmoid activation function is usually used in multilayer NNs that are trained using the backpropagation learning algorithm [28].

5.1.2. Hard limit activation function

Artificial neurons that contain this function can classify the inputs into two different categories. The implemented hard limit activation function follows the next expression:

$$y = \begin{cases} 1 & if \quad net \geq 0 \\ 0 & if \quad net < 0 \end{cases} \tag{3}$$

5.1.3. Symmetrical hard limit activation function

In the symmetrical hard limit activation function, if the function argument is less than 0 then the output of the neuron is -1, if the function is greater than or equal to 0 then the output of the neuron is 1. Symmetrical hard limit function can be defined as follows:

$$y = \begin{cases} 1 & if \quad net \geq 0 \\ -1 & if \quad net < 0 \end{cases} \tag{4}$$

5.1.4. Saturating linear activation function

This type of nonlinear activation functions is also referred to as piecewise linear function. It has either a bipolar or binary range for the saturation limits of the output. The symmetric saturation function is described as follows:

$$y = \begin{cases} 1 & if \quad net > 1 \\ net & \\ 0 & if \quad net < 0 \end{cases} \tag{5}$$

5.1.5. Symmetrical saturating linear activation function

In the symmetrical saturating linear activation function, if the function argument is less than -1 then the output of the neuron is -1, if the function is greater than 1 then the output of the neuron is 1, if the function is greater than or equal to 1 and less than or equal -1 then the output of the neuron is as the same as the input. The symmetrical saturating function is described as follows:

$$y = \begin{cases} 1 & if \quad net > 1 \\ net & if \quad -1 \leq net \leq 1 \\ -1 & if \quad net < -1 \end{cases} \tag{6}$$

5.1.6. Positive linear activation function

This function returns the output of the neuron to input if the function argument is greater than or equal to 0, if the function is less than 0 then the output of the neuron is 0.

$$y = \begin{cases} 0 & if \quad net < 0 \\ net & if \quad 0 \leq net \end{cases} \tag{7}$$

All the aforementioned activation functions can be summarized pictorially in Fig. 6.

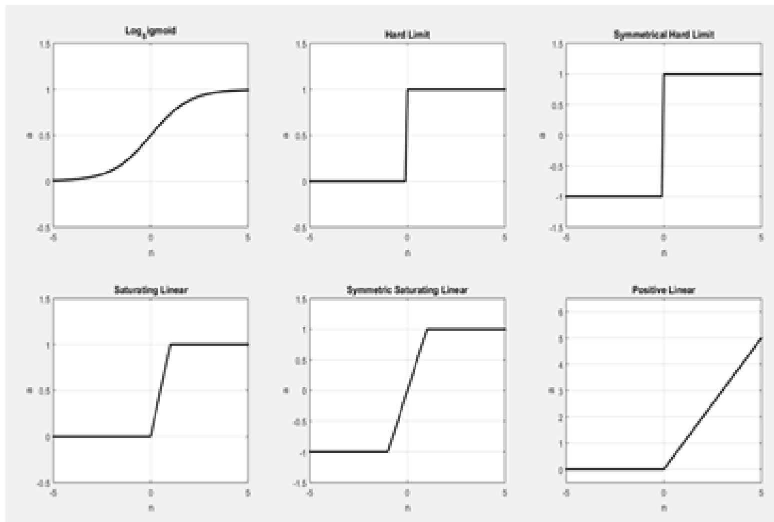


Fig. 6. The waveforms of Sigmoid, hard limit, symmetrical hard limit, saturating linear, symmetrical saturating linear and positive linear activation functions.

6. Implementation results

The XOR problem can be illustrated in Fig. 7. The implemented FFNN architecture for solving this problem consists of 3 layers (input, hidden, and output layers), and 2 hidden neurons. The implemented FFNN architecture also is called Multilayer perceptron (MLP) model as shown in Fig. 8. In the context of implementation, the following approach was followed:

1. Initializing MLP and assign random values [-1, 1] to the weights.
2. Calculating the weighted sum of the inputs at the neurons of hidden layer using sigmoid activation function in backpropagation learning algorithm.
3. Running the calculated value through different activation functions.
4. Following the same steps for the output layer using the outputs of the hidden layer as inputs for the output layer.
5. Calculating deltas values for the output and hidden layer.
6. Updating the weights using the delta values.
7. The training parameter (η) = 0.5.

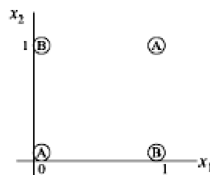


Fig. 7-a. XOR problem: points (0,0) and (1,1) belong to class A; points (0,1) and (1,0) belong to class B.

Inputs		Output
in_0	in_1	result1
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 7-b. The Truth table for logical- XOR problem.

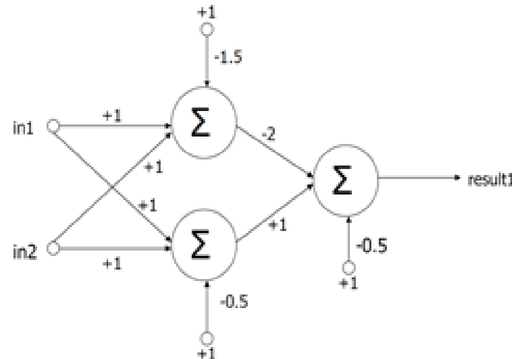


Fig. 8. MLP model for XOR Problem.

In this section five FPGA-based FFNNs will be coded using VHDL and implemented using Digilent Basys 2 Spartan-3E FPGA platform from Xilinx. The Basys 2 board is a circuit design and implementation platform that can be used to build real digital circuits. Basys2 is constructed around a 100,000-gate Xilinx Spartan-3E FPGA and an Atmel "AT90USB2" microcontroller. Basys 2 board is shown in Fig. 9.

The implemented activation functions are hard limit, symmetrical hard limit, saturating linear, symmetrical saturating linear and positive linear activation functions.

Once the design is coded, the top module is synthesized and verified for its timing and functionality using Xilinx ISE (Integrated Synthesis Environment) Design Suite (ver.14.7) software tool. Fig. 10 displays the generated RTL schematic of the coded NN model for XOR problem. Once the process of timing and functionality verifications are completed and the bit-stream file is generated then this file will be downloaded to FPGA to implement the software design in hardware.

Fig. 9. The Basys 2 board.

Our implementations follow the chip-in-the-loop learning mechanism which entails both the hardware network for carrying out the computations and Back Propagation (BP) learning algorithm for updating the NN weights. Sigmoid activation function was utilized by backpropagation learning algorithm to get weights and biases values which are as follow b_1= 0010000, w_{10} = 1101000, w_{11}= 0010000, w_{12}= 0010000, b_2 = 0010000, w_{20} = 1111000, w_{21}= 0010000, w_{22} = 0010000, b_3 = 0010000, w_{30} = 1111000, w_{31} = 1100000, w_{32} = 0010000. These values were used through the examination of the introduced five nonlinear activation functions.

For each implemented ANN, set of (64) input pairs [in1, in2] combinations were used for testing, where each of in1 and in2 contains 3-bits. The main of each implemented ANN is to predict the correct response for each input combination.

JES, Assiut University, Faculty of Engineering, Vol. 46, No. 5, September 2018, pp.539–555

Table 2 exhibits the implementation utilization for each implemented ANN, where each one uses one of the five examined activation functions. Table 3 displays the performance comparison between examined activations functions in terms of correct predictions percentages for XOR problem.

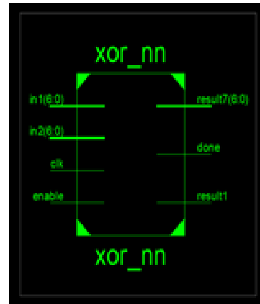


Fig. 10. The RTL schematic of the MLP model for XOR problem at using any of the implemented activation functions. (RTL: register transfer logic)

Table 2.
Implementation Utilization Summary of the Examined Activation Functions for XOR Problem.

Logic Utilization	Utilization				
	Saturating Linear	Hard Limit	Symmetrical Saturating Linear	Symmetrical Hard Limit	Positive Linear
Number of Slice Flip Flops	7%	6%	7%	6%	8%
Number of occupied Slices	15%	14%	16%	14%	17%
Number of Slices containing only related logic	100%	100%	100%	100%	100%
Total Number of 4 input LUTs	12%	11%	13%	11%	14%
Number of BUFGMUXs	4%	4%	4%	4%	4%

Table 3.
Performance comparison between examined activations functions in terms of correct and incorrect predictions percentages for XOR problem.

Activation Function	Correct Prediction Percentage
Saturating Linear	87.5%
Hard Limit	84.357%
Symmetrical Saturating Linear	62.5%
Symmetrical Hard Limit	60.937%
Positive Linear	56.25%

It is clear from the tabulated results that the saturating linear activation function outperforms all examined activation functions by 87.5% correct prediction percentage, while its implementation occupies 15% of the available slices on the used FPGA platform. Also, it is notable the superiority of the hard limit activation function which provides 84.357% of predictions are correct and occupies 14% of the available slices on the used FPGA platform. The rest of the examined activation functions provide poor performance in comparison with the aforementioned peers. Also, it is clear from Table 2 that all implemented FPGA-based FFNNs have reasonable implementation areas in range from 14% to 17%.

Neural Networks can be used efficiently in simulating the behavior of the digital circuits, and hence the various related applications. The behaviors of digital circuits are required to be implemented in hardware form as solutions to specific problems. In order to study the workability, and reliability of FPGA-based ANNs, another case will be investigated, which is Full-Adder circuit.

A logic circuit for a full-adder is shown in Fig.11, and the truth table in Table 4 shows the basic operation of a full-adder. The implemented FFNN architecture for solving full-adder problem consists of 3 layers (input, hidden, and output layers), and 3 hidden neurons as shown in Fig.12. The generated RTL schematic of the coded NN model for full-adder problem is shown in Fig.13. The weights and biases values for full-adder problem can be summarized as follow: b_1= 0010000,w_{10} = 0000101, w_{11}= 0110010, w_{12}= 1101010, w_{13}=0110001, b_2= 0010000, w_{20} = 0000000, w_{21}=1001000, w_{22}= 0001100, w_{23}= 1001000, b_3= 0010000, w_{30}= 1101100, w_{31}= 0000000, w_{32}= 1101011, w_{33}= 0000000, b_4= 0010000, w_{40} = 1110010, w_{41} = 0010101, w_{42}= 1101110, w_{43}= 1100101, b_5= 0010000, w_{50} = 0000000, w_{51} = 1111101, w_{52}= 0010011, w_{53} = 0000000.

Table 5 introduces the performance comparison between examined activations functions in terms of correct predictions percentages for full-adder problem. The introduced results in Table 5 demonstrate the superiority of both saturating linear and hard limit activation functions over the rest of examined activation functions.

Implementation utilization summary of all examined activation functions for full-adder Problem is given in Table 6. It is clear that all implemented FPGA-based FFNNs have good implementation areas in range from 49% to 52%.

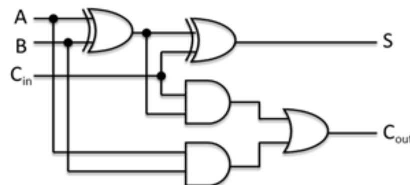


Fig. 11. Logic circuit for a full-adder

Table 4.
Full-Adder truth table.

Inputs			Output	
A	*B*	C_{in}	C_{out}	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

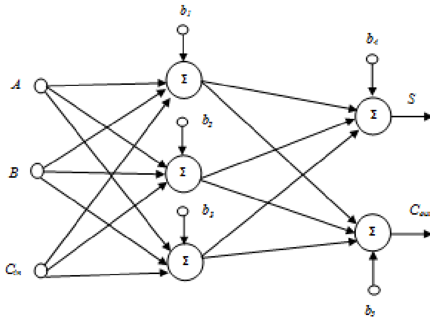


Fig. 12. MLP model for Full-Adder Problem

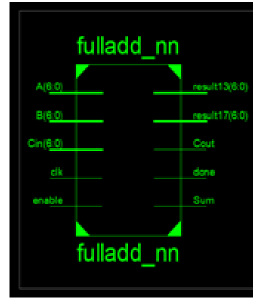


Fig. 13. RTL schematic of the MLP model for Full-Adder Problem

Table 5.

Performance comparison between examined activations functions in terms of correct and incorrect predictions percentages for full-adder problem.

Activation Function	Correct prediction percentage for C_{out}	Correct prediction percentage for S
Saturating Linear	83.5%	86.5%
Hard Limit	80.5%	82.5%
Symmetrical Saturating Linear	64%	64.5%
Symmetrical Hard Limit	60.5%	62%
Positive Linear	59.5%	59.5%

Table 6.

Implementation Utilization Summary of the Examined Activation Functions for Full-Adder Problem.

Logic Utilization	Utilization				
	Saturating Linear	Hard Limit	Symmetrical Saturating Linear	Symmetrical Hard Limit	Positive Linear
Number of Slice Flip Flops	23%	22%	23%	22%	24%
Number of occupied Slices	50%	49%	51%	49%	52%
Number of Slices containing only related logic	100%	100%	100%	100%	100%
Total Number of 4 input LUTs	44%	43%	45%	42%	46%
Number of BUFGMUXs	4%	4%	4%	4%	4%

The most vital component of a neuron is the activation function. In the FPGA-based ANNs, it is not so easy to implement the most commonly used sigmoid activation functions, because it consists of an infinite exponential series and this is considered as a difficult challenge faced by designers [29]. So, the designers try to deduce approximations for sigmoid functions to be used with FPGA designs.

There are many practical approaches to approximate sigmoid functions, such as Piece-wise linear approximation that describes a combination of lines in the form of y=ax+b which is used to approximate the sigmoid function; and lookup tables, in which uniform samples taken from

the centre of a sigmoid function can be stored in a table for look up. The regions outside the centre of the sigmoid function are still approximated in a piece-wise linear fashion [30].

Our main contribution is to introduce computationally simplified alternatives of sigmoid function, which can be directly implemented using FPGA, and at the same time offer an area-efficient implementation. The examined activation functions are already known in software-based ANNs, but are not commonly used; also they are not used in hardware-based ANNs.

The obtained results are satisfactory and give a clear view of the parallel processing capabilities of FPGAs. These capabilities can be exploited to construct robust ANNs, which in turn can be used efficiently in real-time applications.

7. Conclusion

In this paper, the XOR and Full-Adder problems were used as benchmarks for comparing the performance of FPGA-based ANNs. The performances of the implemented NNs were investigated in terms of area efficient implementation, and correct prediction percentage using five different activation functions.

Both saturating linear and hard limit activation functions aided to build two efficient ANNs which are achieved superior performance and support designs with reasonable implementation areas.

Where the implemented FPGA-based ANNs contain only one hidden layer that contains only 2 (for XOR problem), or 3(for full-adder problem) hidden neurons, exceptional consideration must be paid to an area-efficient implementation when implementing large FPGA-based ANNs.

8. Future work

Our Future research should further build and investigate another NNs types such as Recurrent NN, Convolutional NN, and Radial Basis Function NN, in order to get the most powerful NN type that adequate for practical implementation of FPGA-based ANNs.

REFERENCES

[1] M. Moussa and S. Areibi and K. Nichols, "On the Arithmetic Precision for Implementing Back-Propagation Networks on FPGA". University of Guelph, school of engineering. Guelph, Ontario, Canada. 2003.

[2] A. Muthuramalingam, S. Himavathi and E. Srinivasan. "Neural Network Implementation Using FPGA: Issues and Application". The International Journal of Information Technology. vol. 4. no. 2. pp.86-92. 2008.

[3] S. Abrol1and Mrs. Rita. Mahajan. "Artificial Neural Networks Implementations on FPGA Chip". Chandigarh, India. Vol. 3, Issue 1, pp: (11-18). 2015.

[4] Rolf F. Molz, Paulo M. Engel and Fernando G. Moraes. "Codesign to Fully Parallel Neural Network for a Classification Problem". University Montpellier II. France. 2000.

[5] K.V. Ramanaiah and S. Sridhar. "Hardware Implementation of Artificial Neural Networks". India. Vol. 3 l No. 4 l November 2014 - January 2015.

[6] M. Stevenson, R. Winter and B. Widrow. "Sensitivity of Feedforward Neural Networks to Weight Errors". IEEE Transactions on Neural Networks. Vol.1. No 2. pp71-80. 1990.

[7] H. Sak, A. Senior and F. Beaufays. "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling". 15th Annual Conference of the International Speech Communication Association. Singapore. September14-18, 2014.

[8] S. Haykin (16 July 1998). "Chapter 2: Learning Processes". Neural Networks: A comprehensive foundation (2nd ed.). Prentice Hall. pp. 50–104. ISBN 978-8178083001. Retrieved 2 May 2012.

[9] David E. Rumelhart, Ronald J. Williams and Geoffrey E. Hinton. “Learning Representations by Back-Propagating Errors". Nature. 323 (6088): 533–536. 8 October 1986.

[10] M. Forssell. "Hardware Implementation of Artificial Neural Networks ". 18-859E Information Flow in Networks. 2014.

[11] L.P. Maguire, T.M. McGinnity, B. Glackin, A. Ghani, A. Belatreche and J. Harkin. " Challenges for Large-Scale Implementations of Spiking Neural Networks on FPGAs". Neurocomputing 71:1-3, 13-29. 2007.

[12] R. Katriel. “Three Highly Parallel Computer Architectures and Their Suitability for Three Representative Artificial Intelligence Problems ". University of Pennsylvania. Philadelphia, PA 19104. February 1988.

[13] D.Walsh. "Design and Implementation of Massively Parallel Fine-Grained Processor Arrays". Manchester, UK: The University of Manchester. 2015.

[14] S.K. Foo, P. Saratchandran, and N. Sundararajan. " Parallel Implementation of Backpropagation Neural Networks on a Heterogeneous Array of Transputers". IEEE Trans. on Systems, Man, and Cybernetics–Part B: Cybernetics, 27(1):118–126, 1997.

[15] M. Schaefer, T. Schoenauer, C. Wolff, G. Hartmann, H. Klar, and U. Ruckert. "Simulation of Spiking Neural Networks - Architectures and Implementations". Neurocomputing, (48):647–679, 2002.

[16] A. Kramer. "Array-Based Analog Computation: Principles, Advantages and Limitations". In Proc. MicroNeuro. Milan, Italy. pages 68–79. 1996.

[17] Y.K. Choi, K.H. Ahn, and S.-Y. Lee. "Effects of Multiplier Output Offsets on On-Chip Learning for Analog Neuro-Chips". Neural Processing Letters, 4:1–8, 1996.

[18] A. Montalvo, R. Gyurcsik, and J. Paulos. "Towards a General-Purpose Analog VLSI Neural Network with On-Chip Learning". IEEE Trans. on Neural Networks, 8(2):413–423, 1997.

[19] W. Eppler, T. Fisher, H. Gemmeke, T. Becher, and G. Kock. "High Speed Neural Network Chip on PCI-Board". In Proc. MicroNeuro. pages 9–17. 1997.

[20] A. Johannet, L. Personnaz, G. Dreyfus, J.D. Gascuel, and M. Weinfeld. "Specification and Implementation of a Digital Hopfield-Type Associative Memory with On-Chip Training". IEEE Trans. on Neural Networks, 3. 1992.

[21] B. Girau. "FPNA: Concepts and Properties". FPGA Implementations of Neural Networks. pp 63-101. 2006.

[22] P.J. Ashenden. "VHDL Standards", IEEE Design & Test of Computers. vol. 18, n. 6. pp. 122–123. September–October 2001.

[23] U. Farooq, Z. Marrakchi, and H. Mehrez. "FPGA Architectures: An Overview". Tree-based Heterogeneous FPGA Architectures. Springer, New York, NY. pp 7-48. 2012.

[24] Basys 2™ FPGA Board Reference Manual. Revised April 8, 2016. Online.https://store.digilentinc.com/basys-2-spartan-3e-fpga-trainer-board-limited-time/

[25] K.M. Sammut and S.R. Jones. "Arithmetic unit design for neural accelerators: cost performance issues". IEEE Trans. on Computers, 44(10). 1995.

[26] Aditya Sharmaa. "Deep Learning, Machine Learning: Understanding Activation Functions in Deep Learning". October 30, 2017. (This post is part of the series on Deep Learning for Beginners).

[27] M. Leshno, V.Y. Lin, A. Pinkus and S. Schocken. "Neural Networks: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". Volume 6. Issue 6. Pages 861-867. 1993.

[28] B. Bateja and P. Sharma. "FPGA Implementation of Artificial Neural Networks". National Institute of Technology, Rourkela. 2007.

[29] Mutlu Avcý, Tulay Yýldýrým, “Generation of Tangent Hyperbolic Sigmoid Function for Microcontroller Based Digital Implementation of Neural Networks”, International XII. Turkish Symposium on Artificial Intelligence and Neural Networks, 2003.

[30] Haitham Kareem Ali, Esraa Zeki Mohammed. "Design Artificial Neural Network Using FPGA". International Journal of Computer Science and Network Security, pp. 88-92, vol.10 No.8, August 2010.

التنفيذ الكفء للمكونات المادية للشبكات العصبية ذات التغذية الأمامية باستخدام مصفوفة البوبات المنطقية القابلة للبرمجة

الملخص العربي

تنفيذ المكونات المادية للشبكات العصبية الاصطناعية يعتمد فى الاساس على تنفيذ دوال التفعيل بكفأه. وتعتبر مصفوفة البوابات بالمنطقية القابلة للبرمجة (FPGA) اداه مناسبة لتنفيذ المكونات المادية للشبكات العصبية الاصطناعية. فى هذه الورقة البحثية نقدم تنفيذ المكونات المادية للشبكات العصبية الاصطناعية مرتكزه على (FPGA) وذلك باستخدام خمس دوال تفعيل مختلفة. وذلك يمكن وصفه من خلال لغة وصف المكونات المادية للدوائر متناهية الصغر ذات السرعات العالية (VHDL) وتنفيذه على شريحة (Digilent Basys 2 Spartan-3E FPGA from Xilinx). تم قياس اداء الشبكات العصبية المصممة من ناحية المساحة المستغلة على شريحة (FPGA) من التصميم وايضا نسبة الصواب والخطأ فى حل هذه الشبكات العصبية المصممة لمشكلتى (XOR) و المجمع الكامل (Full-Adder).