BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Volumul 63 (67), Numărul 3, 2017
Secția
ELECTROTEHNICĂ. ENERGETICĂ. ELECTRONICĂ

DIGITAL NEURON ARCHITECTURE FOR MULTILAYER NEURAL NETWORKS HARDWARE IMPLEMENTATION

BY

ȚIGĂERU LIVIU[*]

"Gheorghe Asachi" Technical University of Iaşi,
Faculty of Electronics, Telecommunications and Information Technology

Received: July 31, 2017
Accepted for publication: August 28, 2017

Abstract. During recent years, neural networks and deep learning had drawn a grown interest from the side of the scientific community and industrial environment, due to their promising potential in solving complex problems. However, the implementation of neural networks for real time applications raises a lot of challenges, caused by the computational burden. The present paper introduces a mixed serial-parallel, scalable digital neuron architecture, suitable for online learning multilayer neural network**s** implementation, able to be configured as a sigmoid or as a hyperbolic tangent neuron, respectively. To test its performance, the proposed neuron was implemented on a set of various FPGA devices. The implementation results indicate it requires less than 1% from the total quantity of the FPGA's logical resources, while its working frequency can be increased over 100 MHz.

Key words: neural networks; neuron model; machine learning; FPGA design.

1. Introduction

Nowadays, machine learning is one of the hottest topics in the information technology field, a broad range of machine learning applications

[*]Corresponding author: *e-mail*: ltigaeru@etti.tuiasi.ro

being reported in scientific journals and magazines. Also, the involvement of the industrial environment in machine learning applications records a strong growing, many important companies such as Google, Nvidia, IBM, Microsoft, Facebook, Apple, etc. investing huge funds to develop their artificial intelligence departments.

One of the most powerful information processing paradigm in machine learning, inspired by the way the biological brain processes the information, is artificial neural network and deep learning. Artificial neural networks use interconnected processing elements, named artificial neurons, to learn patterns from sets of examples, to solve different type of problems such as classification, clusterization, predictive analysis, etc. There are a lot of solutions for artificial neural networks implementation, based of software or hardware approaches. Software solutions provide effective environments for analysing the tackled problem, supplying a rich set of useful tools. However, due to the huge computational requirements, such solutions could exhibit significant limitations in complex, real time applications, such as deep learning neural networks for example. To solve these challenges, scientific community along with important chip companies strive to develop effective support to run these applications. Thus, Nvidia (Nvidia Titan V) and ARM (MALI G72) provide powerful GPUs (*Graphical Processing Units*) as a response to the needs of the artificial neural networks. On the other hand, giants as Intel (Intel Nervana), Google (Tensor Processing Unit), IBM (Hsu, 2016), Hewlett Packard Enterprise (Hemsoth, 2017), come with a hardware perspective, providing dedicated neural chips.

The present work focuses on the processing element of the neural network – the artificial neuron. There are many hardware neuron models reported in the scientific literature. Some of them follow the multilayer perceptron model (Minsky *et al*., 1969), others the Hodgkin - Huxley neuron (Hodgkin *et al*., 1952; Izhikevich, 2003), which exhibits a closer behaviour to the biological counterpart. For each case, there are reported various hardware solutions using analog (Binas *et al*., 2016; Indiveri *et al*., 2015; Neftci *et al*., 2011; Rangan *et al*., 2010; Wijekoon *et al*., 2008; Indiveri, 2006; Cauwenberghs, 1997), digital technologies (Jimenez-Fernandez *et al*., 2016; Matsubara *et al*., 2013; Gompertz *et al*., 2011; Ţigăeru *et al*., 2011; Guangxing *et al*., 2010; Ţigăeru, 2009; Ros *et al*., 2007; Schoenauer *et al*., 2002) and mixed analog-digital technologies (Wijekoon *et al*., 2012; Schemmel *et al*., 2010; Heittmann *et al*., 2002). Each approach has benefits and drawbacks, as structure simplicity and power efficiency for the analogue ones, respectively robustness and memory support for the digital solutions. In the present paper, it is proposed a digital scalable serial-parallel architecture for an artificial neuron, suitable for online learning multilayer neural networks implementations. The mathematical model of the proposed neuron is discussed in section II, followed in section III by a detailed description of its architecture. The section IV concerns the functional verification and the implementation of the neuron. For this purpose, a set of various FPGA devices were considered, to obtain an extended view about

the implementation requirements and performance of the proposed neuron. Final comments about the present work are presented in Section **V**.

2. The Mathematical Model of the Artificial Neuron

The model of the artificial neuron is depicted in Fig. 1. It follows the perceptron model that simulates the basic behaviour of the biological neuron in a simplified manner.

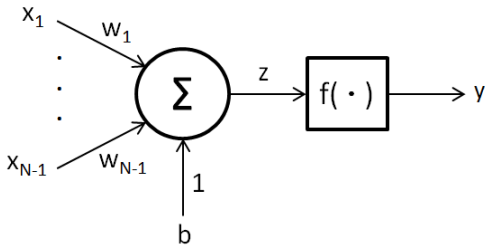

Fig. 1 – The model of the artificial neuron.

According to this, the artificial neuron gathers to its inputs the neural activity produced by the surrounding neurons and computes a net input denoted as *z*, as a weighted sum of its inputs,

$$z = b \cdot 1 + \sum_{i=1}^{N-1} w_i x_i \,, \tag{1}$$

where: b is the neuron's bias and gives a measure of how easy a neuron fires its neural response and w_i – the weight of each neural activity x_i, fired by the $N-1$ neurons to which it is connected.

Subsequently, the net input z is supplied to an activation function $f(\cdot)$, which is used to compute the response of the artificial neuron,

$$y = f(z). \tag{2}$$

The activation function models the firing rate of the electrical impulses generated by the biological neuron, which increases proportionally to its net input. There are many activation functions for the artificial neurons, but the most common ones, used especially in classification applications, are the sigmoid function (3) and the hyperbolic tangent function (4), respectively:

$$f(z) = \frac{1}{1 + \exp(-z)}, \tag{3}$$

$$f(z)=\frac{1-\exp(-2\cdot z)}{1+\exp(-2\cdot z)}. \tag{4}$$

3. The Architecture of the Artificial Neuron

The proposed architecture of the artificial neuron is tailored to be used with the back-propagation error learning algorithm (Rumelhart *et al.*, 1986). The architecture is presented in Fig. 2, where it is considered the case of a neuron with 16 inputs $x_0 \ldots x_{15}$, which could cover a large number of applications. The proposed architecture is scalable, neurons with a larger number of inputs being able to be implemented by expanding the selection block *selBlock*.

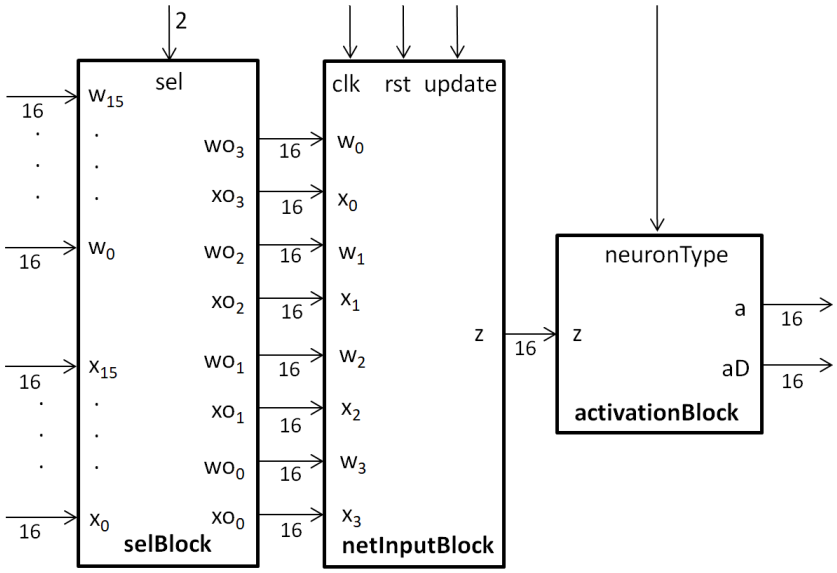

Fig. 2 – The architecture of the artificial neuron

The neuron's inputs takes the neural information $x_0 \ldots x_{15}$ from the connected neurons, weight them with the individual weights $w_0 \ldots w_{15}$ and fires the response y according to the equation:

$$y = f\left(\sum_{i=0}^{15} w_i \cdot x_i\right), \quad \text{where} \quad x_0 = b; \quad w_0 = +1, \tag{5}$$

where the activation function $f(\cdot)$ can be selected between sigmoid function and hyperbolic tangent function, depending on the *neuronType* input state.

Besides the neuron's output *y*, used in the forward phase of the learning algorithm, the architecture also provides its derivative, denoted *yD*, used in the backward phase of the learning algorithm.

All numerical values wherewith the artificial neuron operates use a signed 3.12, fixed point *S_III.FFFFFFFFFFFF* format, where *S* stands for the sign bit, *III* field represents the integer part, and *FFFFFFFFFFFF* field represents the fractional part of the value, respectively. The adopted format confines the numerical values of each neuron's variable (*x, w, z, y*) to the [–8.0 ÷ ÷ +7.999755859375] range, with a precision that ensures the convergence of the back-propagation error learning algorithm (Holt *et al*., 1993).

The proposed architecture consists of three distinct blocks, which implement distinct phases in the computational chain of the neuron's output. The first block, called *selBlock* selects the inputs x_i and theirs corresponding weights w_i in sequentially groups of 4 input-weight pairs, controlled by the *sel* input value. Each selected group of pairs is then supplied in parallel to the *netInputBlock*, which accumulates their sum $x_i \cdot w_i + x_{i+1} \cdot w_{i+1} + x_{i+2} \cdot w_{i+2} + + x_{i+3} \cdot w_{i+3}$ and finally updates the net input *z* value. In the end, the net input *z* is taken by the *activationBlock* which computes the outputs *y* and *yD*, according to the selected activation function.

Subsequently, a detailed description of the main blocks of the neuron architecture is given.

3.1. The Selection Block

The structure of the selection block is based on 4 data inputs multiplexers *mux4* and is depicted in Fig. 3 *a*. If the application requires a larger number of the neuron's inputs, this block can be augmented with another layer of 4 data inputs multiplexers, placed in a tree configuration, keeping the number of selection block's outputs equal to the number of net input block's inputs.

3.2. The Net Input Block

Its inputs number is equal to the number of input-weight pairs, selected by the selection block, which is limited to 4 to accelerate the computation time of *z*. However, for neurons with a large number of inputs, an increase of the inputs number of the net input block has to be taken into a consideration, to balance the propagation times between this and the selection block.

The net input block uses a set of arithmetic circuits which perform signed fixed point multiplications (*multiplier*) and additions (*adder*) respectively, which are placed in a tree structure, as is presented in Fig. 3 *b*. Each *multiplier* truncates the multiplication result to 16 bits, preserving from the 32 bits of the multiplication result the sign bit (the MSB bit), [26:24] field of bits for the integer part and [23:12] field of bits for the fractional part, respectively. Also, to avoid the overflow errors occurring during the computation, each *adder* includes a saturation circuit that bounds its output value to [–8.0 ÷ +7.999755859375].

In the output side, *netInputBlock* has an accumulator structure consisting in *adder* – *rpp* (where *rpp* is a parallel data register), to accumulate the partial sums $x_i \cdot w_i + x_{i+1} \cdot w_{i+1} + x_{i+2} \cdot w_{i+2} + x_{i+3} \cdot w_{i+3}$, and after that, when all groups of input – weight pairs were delivered, to update the *z* value according to the below equation:

$$z = \sum_{j=0}^{3} s_j \quad \text{where} \quad s_j = \sum_{i=4 \cdot j}^{4j+3} w_i x_i \,. \tag{6}$$

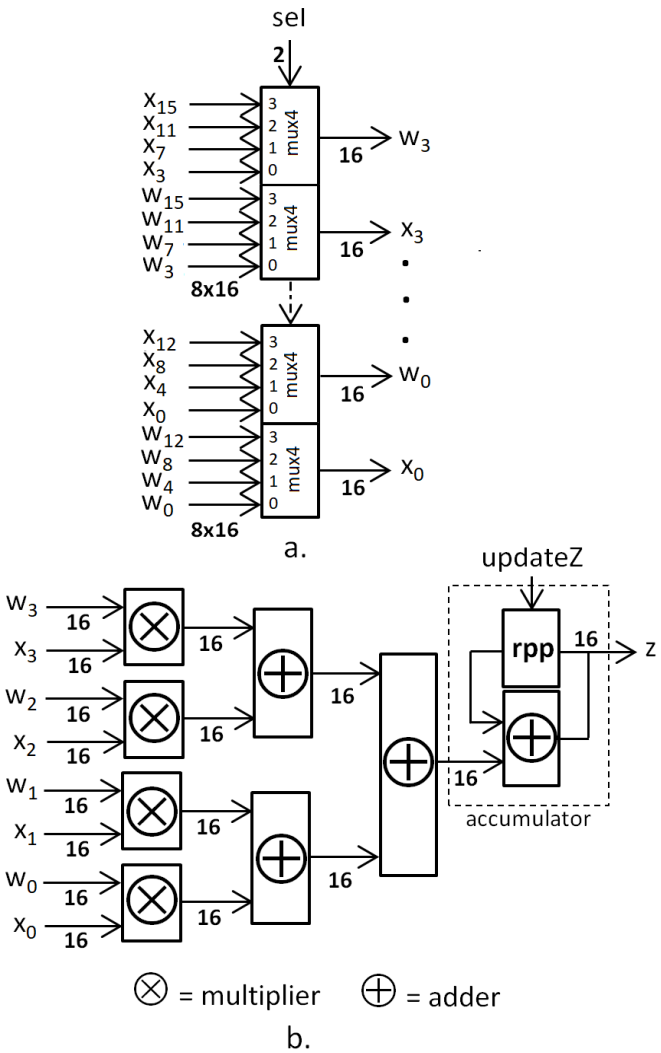

Fig. 3 – *a* – The *selBlock* structure; *b* – the *netInputBlock* structure (for the sake of simplicity, the clock and reset inputs of the register are omitted).

3.3. The Activation Block

The activation block is presented in Figure 4. It consists of 3 main circuits, namely *sigmaCircuit*, which computes the sigmoid function, *sigmaDCircuit*, which computes its derivative and *tanhCircuit*, which computes the hyperboidal tangent function. Besides, *activationBlock* uses a 16 bits adder to compute the derivative of the hyperbolic tangent and two selection circuits, to deliver at its outputs the activation function and its derivative values, respectively, depending on the *neuronType* signal value. Thus, if *neuronType* = 0, then y = sigma(z) and yD = sigmaD(z) and if *neuronType* = 1, then y = tanh(z) and yD = tanhD(z), where the suffix denoted by D suggests the derivative value.

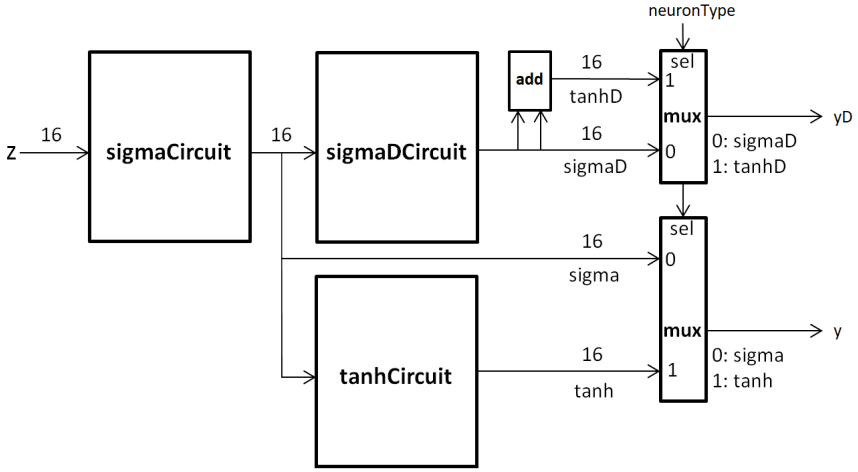

Fig. 4 – The *activationBlock* structure.

There are many approximation methods used to implement the sigmoid function (Tommiska, 2003), of which the one adopted in this paper is that based on PLAN approximation Eq. (7a) (Amin *et al.*, 1997). This method gives a good compromise between the consumed resources and precision (Vassiliadis *et al.*, 2000), according to that, the average error and maximum error, using PLAN method to approximate sigmoid function is 0.59% and 1.89% of the [–8,8] input range, respectively.

The structure of the *sigmaCircuit* is presented in Fig. 5. The two terms added in each PLAN equation are provided by *shifter* circuit and *mux4* multiplexer, respectively, based on the *condition* signal, which is generated by the *conditionDetector* circuit. This circuit is used to identify the range where the absolute value of z belongs. The *shifter* circuit uses the equivalence between the division with power of two and the left shifting operations, to implement the multiplication operation, between $|z|$ and the corresponding PLAN coefficients, which are nothing else but the unit value divided by various powers of two ($0.25 = 2^{-2}$, $0.125 = 2^{-3}$, $0.03125 = 2^{-5}$). As an exception, if the condition $|z| > 5$ is detected, the *shifter* outputs a null value.

One comment has to be done about the representation of the unity value in the proposed architecture. Because it was adopted a 3.12 fixed point format to represent all numerical values, some limitations are introduced in representation of the numerical values. This affects the unity value representation, where the closest value to it is 0.999755859375.

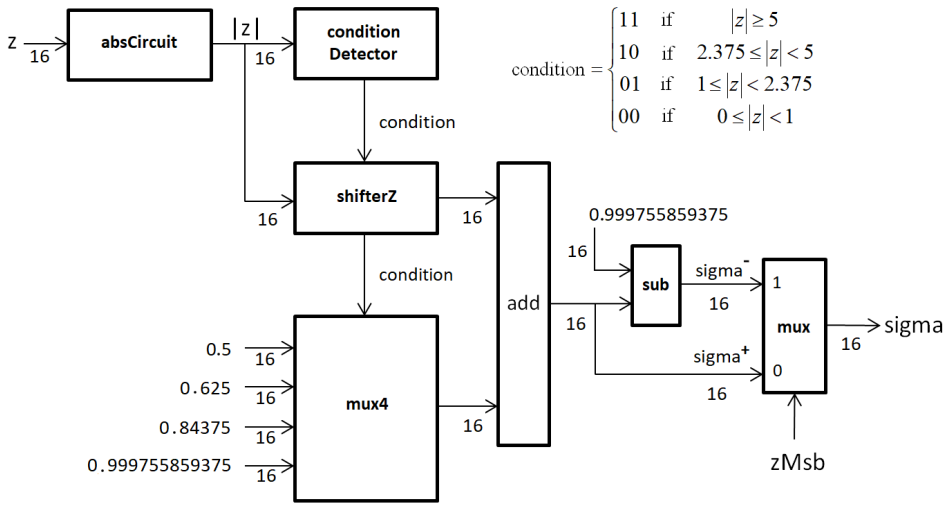

Fig. 5 – The *sigmaCircuit* structure.

Because the PLAN equations provided in Eq. (7a) are valid only for positive values of *z*, a 16 bits subtractor is involved on the output side of *sigmaCircuit*, to implement sigma function for negative values of *z*. In this case, the sigmoid function has to be computed according to the Eq. (7b). The *mux2* multiplexer decides the output value of the sigmoid function, depending on the sign bit of *z* Eq. (7c).

$$sigma^{+} = \begin{cases} 1 & \text{if} \quad |z| \geq 5 \\ 0.03125 \cdot |z| + 0.84375 & \text{if} \quad 2.375 \leq |z| < 5 \\ 0.125 \cdot |z| + 0.625 & \text{if} \quad 1 \leq |z| < 2.375 \\ 0.25 \cdot |z| + 0.5 & \text{if} \quad 0 \leq |z| < 1 \end{cases} \tag{7a}$$

$$sigma^{-} = 1 - sigma^{+} \tag{7b}$$

$$sigma = \begin{cases} sigma^{+} & \text{if} \quad z \geq 0 \\ sigma^{-} & \text{if} \quad z < 0 \end{cases} \tag{7c}$$

The sigmoid function has some nice mathematical properties, which are exploited in this paper, to build *sigmaDCircuit* and *tanhCircuit*, respectively.

First property gives the derivative of the sigmoid function, which can be computed directly from sigmoid function, according to:

$$sigmaD = sigma \cdot (1 - sigma) \tag{8}$$

Second property allows the hyperbolic tangent function to be expressed on the base of sigmoid function:

$$\tanh = 2 \cdot sigma - 1 \tag{9}$$

And finally, the derivative of the hyperbolic tangent function can be expressed using the derivative of the sigmoid function as (10):

$$\tanh D = 2 \cdot sigmaD \tag{10}$$

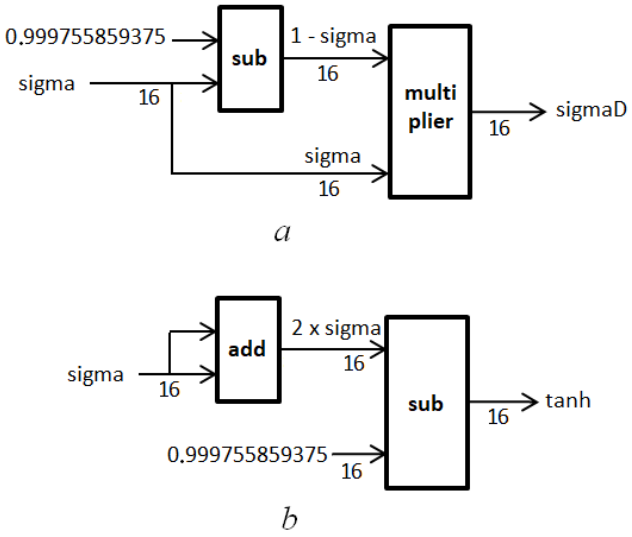

Fig. 6 – *a* – The *sigmaDCircuit* structure; *b* – the *tanhCircuit* structure.

These properties stand as foundation to build the *sigmaDCircuit* presented in Fig. 6 *a*, where *mutliplier* has the same characteristics as the ones previously presented and *tanhCircuit* presented in Fig. 6 *b*, respectively and also explain the presence of the adder circuit denoted by *add*, in the structure of *activationBlock*, depicted in Fig. 4.

4. Results

To validate the functionality of the artificial neuron, a set of verifications of the proposed architecture were performed. For this purpose, the

proposed neuron was modeled in Verilog and implemented on XC7A100T-3CGS324C - Artix 7 Xilinx FPGA device, using Vivado Design Suite software design environment, produced by Xilinx.

The first checks were targeted on the functionality of the activation block, which uses an approximation method to compute the sigmoid and hyperbolic tangent activation functions, and their derivatives, respectively. To this end, the post-synthesis simulation results, generated at the outputs of the activation block, were saved in a text file to be compared with similar results generated by Matlab models of these function, which were considered as ideal ones.

The obtained results are plotted in Fig. 7, for sigmoid function and Fig. 8, for hyperbolic tangent function, respectively. As can be seen, there are small differences between the ideal functions and the ones which are generated by the activation block, which lead to absolute errors a little smaller than 0.02, enough to not undermine the functionality of the proposed neuron.

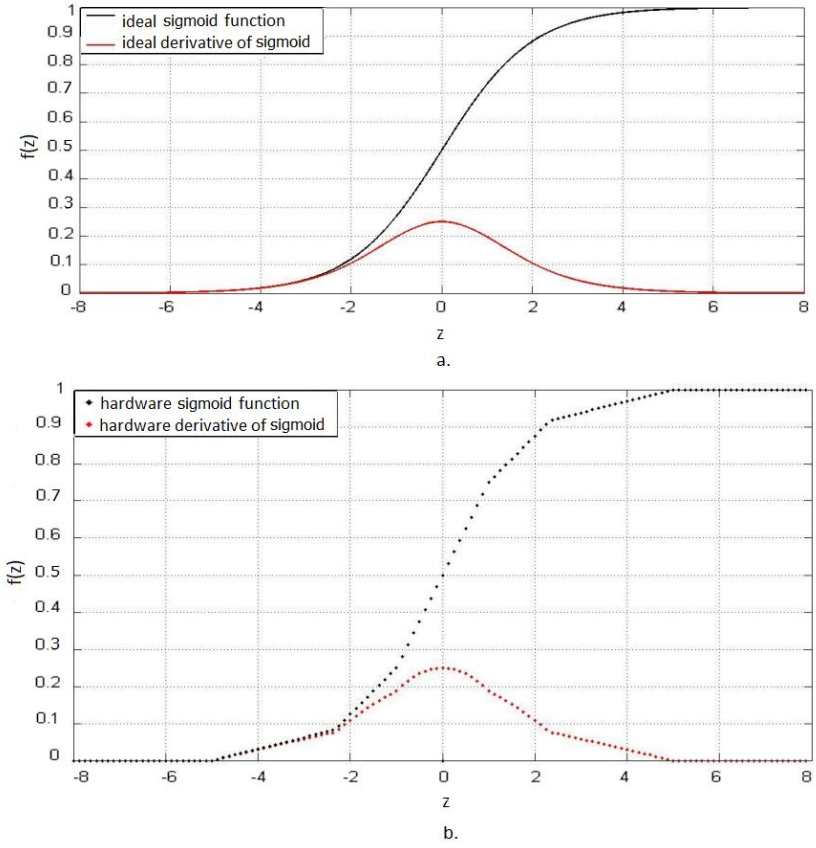

Fig. 7 – The comparison between the ideal and the hardware generated sigmoid functions: *a* – The ideal sigmoid function and its derivative; *b* – the sigmoid function and its derivative generated by the proposed neuron.

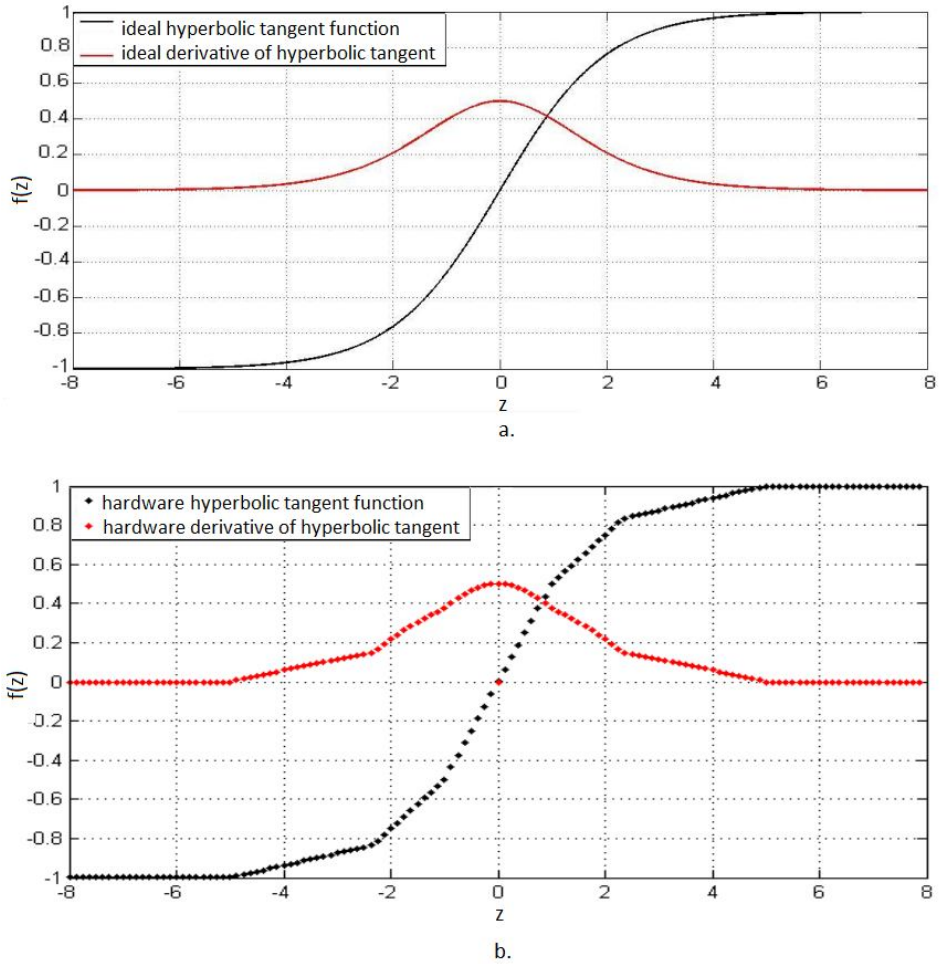

Fig. 8 – The comparison between the ideal and the hardware generated hyperbolic tangent functions: *a* – the ideal hyperbolic tangent function and its derivative; *b* – the hyperbolic tangent function and its derivative generated by the proposed neuron.

The last comment is proved by the next set of investigations, which compare the behaviour of the proposed neuron against an ideal Matlab based neuron model, in two testing scenarios, namely the sigmoid neuron and the hyperbolic tangent neuron, respectively. In this stage of the verification procedure, it were considered a 16 inputs neuron model and 50 sets of random generated inputs-weights (**x**, **w**) pairs, where $\mathbf{x} = (x_0, x_1, \ldots, x_{15})$, and $\mathbf{w} = (w_0, w_1, \ldots, w_{15})$. Each inputs-weights pair was successively delivered to the inputs of both neuron's models, after that their outputs were monitored. The obtained errors' graphs, computed as the absolute difference between the hardware neuron's outputs and the Matlab neuron's outputs, are plotted in the Fig. 9, for

the sigmoid activation function, and in the Fig. 10, for the hyperbolic tangent function. The maximum values of these errors are summarized in Table 1, where *εy* is the error recorded at the *y* output, and *εyD* is the error recorded at the *yD* output. It can be seen that the errors for the hyperbolic tangent are larger than the ones generated for the sigmoid function, due to the hyperbolic tangent function is generated using as support the sigmoid function, which leads to an error accumulation process.

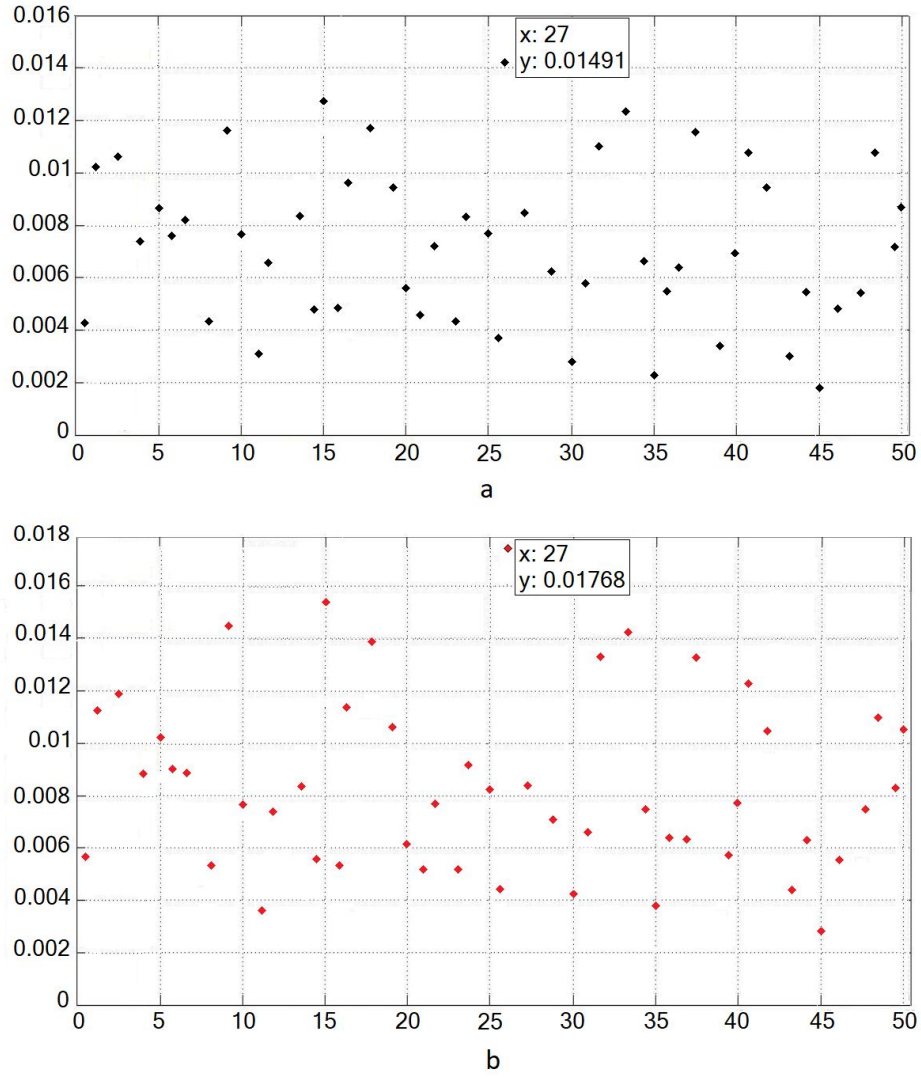

Fig. 9 – The errors of the neuron with sigmoid activation function: *a* – the graph of the error recorded at *y* output; *b* – the graph of the error recorded at *yD* output.

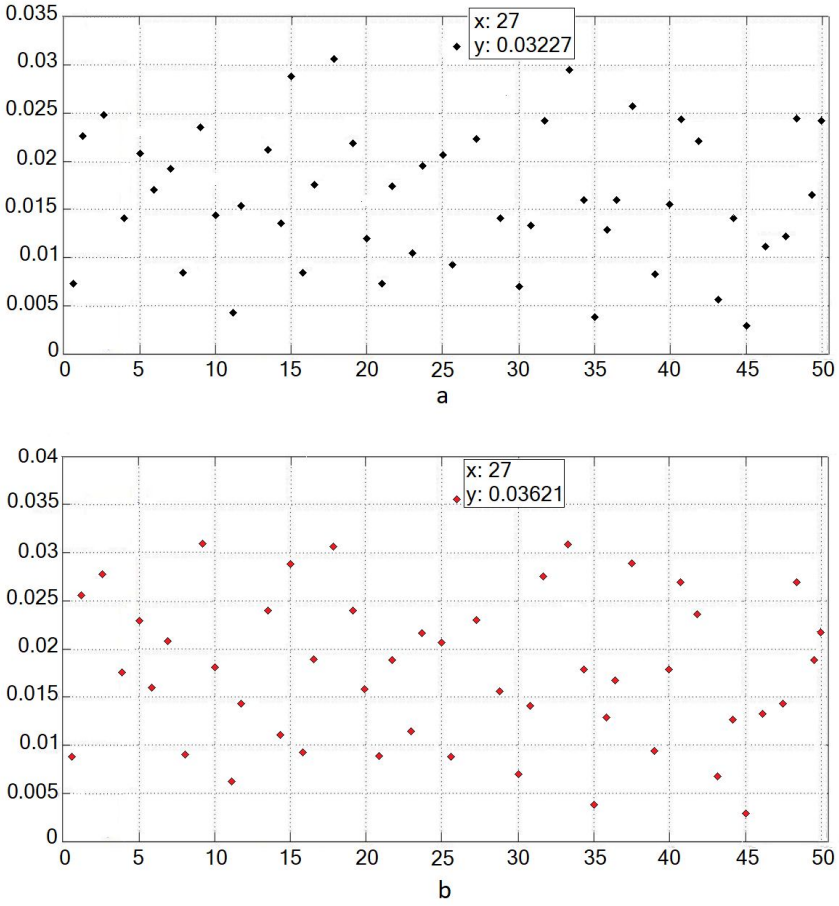

Fig. 10 – The errors of the neuron with hyperbolic tangent activation function: *a* – the graph of the error recorded at *y* output; *b* – the graph of the error recorded at *yD* output.

Table 1
The errors of the proposed neuron

Case	εy	εyD
Sigmoid function	0.01491	0.01768
Hyperbolic tangent function	0.03227	0.03621

Finally, the functionality of the proposed neuron was verified into an application where a multilayer neural network is used to solve a classical xor problem. In the considered test, each neuron use only 3 of 16 inputs (one for bias, two to receive neural information) and the structure of the neural network consists of two neurons on the input layer, 2 neurons on the hidden layer and one neuron on the output layer. Besides the investigated neurons, the architecture of the neural network contains another computational blocks, to

support online back-propagation learning algorithm, but the description of these is beyond the goal of the present paper, being a topic for a future one.

The neural network was trained, and then validated, in two distinct scenarios. For the first one, it was considered neurons with sigmoid activation function. In the second one, it was considered neurons with hyperboidal tangent activation function. The neural network was implemented in the same FPGA device as the one previously used and the results were saved and plotted in Fig. 11. As can be seen, the results confirm the expected behaviour of the neural network, which is able to learn and solve the xor problem after a number of training epochs. For the considered problem, the hyperboidal tangent activation function seems to offer a better solution than the sigmoid alternative, leading to a faster learning process.

Consequently, all these results validate the functionality of the proposed artificial neuron.

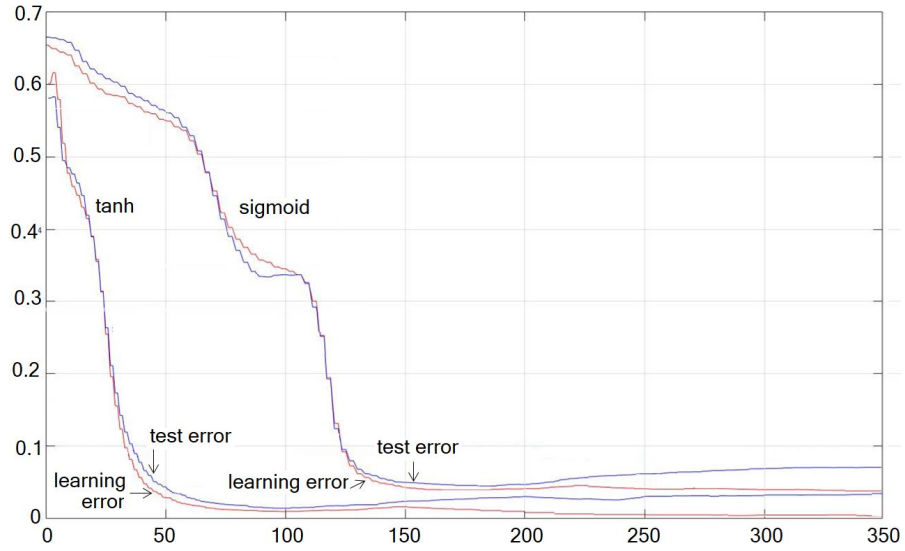

Fig. 11 – The results recorded by the neural network for solving xor problem (350 epochs).

Another concern of the present work was the implementation performances of the proposed neuron on various FPGA devices, including MPSoC FPGA type. For this purpose, it was considered a 16 inputs neuron, implemented on a set of Xilinx FPGA devices, on which only the ones that gave the most representative results are enumerated in Table 2, which summarize the consumed logical resources for neuron's implementation and the maximum working frequency, respectively.

In each considered case, the implementation settings were kept to the default values of Vivado Design Suite design environment. Consequently, the

reported results are not the most optimistic ones, further improvements could being achieved by using another implementation settings and introducing timing constraints for critical paths.

Table 2
Implementation Results

FPGA Device	Consumed Resources	f_{MAX} [MHz]	f_{MAX} pipeline [MHz]
Kintex UltraScale + xcku5p – ffvb676 – 3 **16nm***	LUTs: 184/216960 (< 1%) DSP Blocks: 5/1824 (< 1%) FF registers: 16/433920 (< 1%)	105.9	128.33
Zynq UltraScale + xczu2cg – sfva625 – 3 **16nm**	LUTs: 184/47232 (< 1%) DSP Blocks: 5/240 (< 2%) FF registers: 16/94464 (< 1%)	98.2	108.43
Kintex UltraScale xcku035 – fbva676 -3 **20nm**	LUTs: 184/203128 (< 1%) DSP Blocks: 5/1700 (< 1%) FF registers: 16/406256 (< 1%)	62.73	91.82
Zynq-7000 xc7z020- 3 **28nm**	LUTs: 230/53200 (<1%) DSP Blocks: 5/220 (< 2%) FF registers: 16/106400 (< 1%)	39.66	54.3
Artix 7 xc7a100T-cgs324-3 **28nm**	LUTs: 230/63400 (< 1%) DSP blocks: 5/240 (< 2%) FF registers: 1 /126800 (< 1%)	36.77	52.79
Spartan 7 xc7s50 – cgs324 – 3 **28nm**	LUTs : 230/32600 (< 1%) DSP blocks: 5/120 (< 4%) FF registers: 16/65200 (< 1%)	32.41	46.26

* technology

Regarding the consumed FPGA's resources, UltraScale family FPGA devices provide better implementation solutions. However, irrespective of the adopted FPGA device, the reported results show that less than 1% of the total logical resources of any considered FPGA devices are consumed to implement the proposed neuron. This suggests there remains enough room to implement medium size neural networks on a single FPGA device, using the proposed neuron. On the other hand, if a large number of neurons is required by the application, a network of multiple interconnected FPGA devices can be considered as an implementation option.

To investigate the maximum working frequency, a neural network with two neurons on the input layer and one neuron on the output layer was implemented on the considered FPGA devices. The maximum clock frequency, to which the considered neural network can operate reliably, is reported on the third column of the Table 2. Once again UltraScale family FPGA devices offer the highest performance, which is close to 100MHz working frequency, while for the other FPGA devices, the performance is about half below. This gap is explained by innovations specific to UltraScale+/UltraScale 16 nm/20 nm

families, which claim to lead to 2÷5X greater system-level performance over other 28 nm FPGA devices.

Additional improvements can be obtained by enforcing timing constrains on critical paths, or by using high end FPGA devices, as Virtex UltraScale+.

A global improvement of the timing performance could be achieved by inserting layers of pipeline registers at the outputs of the selection block and activation block, respectively. This technique was explored on the proposed neuron and the results are reported in the last column of Table 2. Once again, timing constrains on critical paths could bring additional performance improvements.

Although the pipeline technique is beneficial regarding the performance of the proposed neuron, the decision of using it has to be correlated with the solution used in neural work to implement the learning algorithm. Without this concern, the overall performance of the implemented neural network can be deteriorated.

5. Conclusion

The present paper introduces a digital neuron scalable architecture, suitable for online learning multi-layer neural networks implementation. Depending on a configuration signal, the neuron model can use one of two types of activation functions, namely sigmoid and hyperbolic tangent respectively, which are implemented based on PLAN approximation method.

The functionality of the neuron model has been extensively verified in various testing scenarios compared with ideal models and the errors were monitored. Their small values confirm the feasibility of the proposed neuron which is proved in a multilayer neural network used to solve an xor problem.

Finally, an analysis of the performance of the proposed neuron model implemented on various Xilinx FPGA devices was performed. The obtained results indicate the working frequency can be raised above 100 MHz, without use of timing constraints. Regarding the implementation resources, the proposed neuron requires less than 1% from the total quantity of the FPGA's logical resources, which allows medium size neural networks to be implemented on a single FPGA device.

REFERENCES

Amin H., Curtis K.M., Hayes–Gill B.R., *Piecewise Linear Approximation Applied to Nonlinear Function of a Neural Network*, IEEE Proc. Circ., Devices Syst., **144**, *6*, 313-317 (1997).

Binas J., Indiveri G. Pfeiffer M., *Spiking Analog VLSI Neuron Assemblies as Constraint Satisfaction Problem Solvers*, Internat. Symp. on Circ. a. Syst. (ISCAS), pp. 2094-2097, 2016

Cauwenberghs G., *Analog VLSI Stochastic Perturbative Learning Architectures*, Analog Integrated Circuits and Signal Processing, **13,** *1-2*, 195-209 (1997).

Gomperts A., Ukil A., Zurfluh F., *Development and Implementation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications*, IEEE Trans. on Industrial Informatics, **7**, *1*, 78-89 (2011).

Guangxing L., Vargha T., Ahmad Y., Curtis L. B. Jr., *A FPGA Real-Time Model of Single and Multiple Visual Cortex Neurons*, J. of Neuroscience Methods, *193*, 62-66 (2010).

Heittmann A., Ruckert U., *Mixed Mode VLSI Implementation of a Neural Associative Memory*, Analog Integrated Circ. and Signal Proc., **30**, *2*, 159-172 (2002).

Hemsoth N., *HPE Developing its own Low Power Neural Network Chips*, https://www.nextplatform.com, 2017.

Hodgkin A.L., Huxley F., *A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve*, J. Physiol., **116**, 507-544 (1952).

Hodgkin A.L., Huxley F., *Currents Carried by Sodium and Potassium Ions Through the Membrane of the Giant Axon of Loligo*, J. Physiol., **116**, 449-472 (1952).

Hodgkin A.L., Huxley F., Katz B., *Measurements of Current–Voltage Relations in the Membrane of the Giant Axon of Loligo*, J. Physiol., **116**, 424-448 (1952).

Hodgkin A.L., Huxley F., *The Components of Membrane Conductance in the Giant Axon of Loligo*, J. Physiol., **116**, 473-496 (1952).

Hodgkin A.L., Huxley F., *The Dual Effect of Membrane Potential on Sodium Conductance in the Giant Axon of Loligo*, J. Physiol., **116**, 497-506 (1952).

Holt J.L., Hwang J-N., *Finite Precision Error Analysis of Neural Network Hardware Implementations*, IEEE Trans. on Comp., **42**, *3*, 281-290 (1993).

Hsu J., *IBM's Brain-Inspired Chip Tested for Deep Learning*, IEEE Sprectrum, 2016.

Indiveri G., Chicca E., Douglas R., *A VLSI Array of Low-Power Spiking Neurons and Bistable Synapses with Spike–Timing Dependent Plasticity*, IEEE Trans. on Neural Networks, **17**, *1*, 211-221 (2006).

Indiveri G., Corradi F., Qiao N., *Neuromorphic Architectures for Spiking Deep Neural Networks*, IEEE Internat.l Electron Devices Meeting (IEDM), 2015.

Izhikevich E. M., *Simple Model of Spiking Neurons*, IEEE Trans. on Neural Networks, **14**, *6*, 1569-1572 (2003).

Jimenez-Fernandez A., Cerezuela-Escudero E., Miro-Amarante L., Domınguez-Morales M. J., de Asıs Gomez-Rodıguez F., Linares Barranco A., Jimenez-Moreno G., *A Binaural Neuromorphic Aditory Sensor for FPGA: a Spike Signal Processing Approach*, IEEE Trans. on Neural Networks and Learning Systems, 2016.

Matsubara T., Torikai H., *Asynchronous Cellular Automaton-Based Neuron: Theoretical Analysis and on-FPGA Learning*, IEEE Trans. on Neural Networks and Learning Syst., **24**, *5*, 736-748 (2013).

Minsky M., Papert S., *Perceptrons: an Introduction to Computational Geometry*, MIT Press, 1969.

Neftci E., Chicca E., Indiveri, G., Douglas R., *A Systematic Method for Configuring VLSI Networks of Spiking Neurons*, Neural Computation, **23**, *10*, 2457-2497 (2011).

Rangan V., Ghosh A., Aparin V., Cauwenberghs G, *A Subthreshold A VLSI Implementation Of The Izhikevich Simple Neuron Model*, Ann. Internat. IEEE Conf. on Engng. in Medicine and Biology Society (EMBC), 4164-4167, 2010.

Ros E., Ortigosa E.M., Agis R., Carrillo R., Arnold M., *Real-Time Computing Platform for Spiking Neurons (RT-Spike)*, IEEE Trans. on Neural Networks, **17**, *4*, 1050-1063 (2007).

Rumelhart D.E., Hinton G.E., Williams R.J., *Learning Representations By Back-Propagating Errors*, Nature., **323**, 533-536 (1986).

Schemmel J., Bruderle D., Grubl A., Hock M., Meier K., Millner S., *A Wafer-Scale Neuromorphic Hardware System for Largescale Neural Modeling*, Internat. Symp. on Circ. a. Syst. (ISCAS, 2010), pp. 1947-1950

Schoenauer T., Atasoy S, Mehrtash N., Klar H., *NeuroPipe-Chip: A Digital Neuro-Processor for Spiking Neural Networks*, IEEE Trans, on Neural Networks, **13**, *1*, 205-213 (2002).

Tommiska M.T., *Efficient Digital Implementation of the Sigmoid Function for Reprogrammable Logic*, IEE Proceedings – Computers and Digital Techniques, **150**, *6*, 403-411 (2003).

Vassiliadis S., Zhang M., Delgado-Frias J.G., *Elementary Function Generators for Neural-Network Emulators*, IEEE Trans. on Neural Networks, **11**, *6*, 1438-1449 (2000).

Wijekoon J.H.B., Dudek P., *Compact Silicon Neuron with Spiking and Bursting Behaviour*, Neural Networks, **21**, 524-534 (2008).

Wijekoon J.H.B., Dudek P., *VLSI Circuits Implementing Computational Models of Neocortical Circuits*, J. of Neuroscience Methods, **210**, *1*, 93-109 (2012).

ARHITECTURA DIGITALĂ PENTRU NEURON DEDICAT IMPLEMENTĂRII HARDWARE A REŢELELOR NEURONALE MULTISTRAT

(Rezumat)

În ultimii ani, reţelele neuronale şi de tip deep learning au atras un interes crescut din partea comunităţii ştiinţifice şi a mediului industrial, datorită potenţialului lor promiţător în rezolvarea problemelor complexe. Totuşi, implementarea acestora pentru aplicaţii în timp real ridică multe probleme, determinate de către cerinţele de calcul. Articolul prezent introduce o arhitectură mixtă, de tip serie-paralel, scalabilă, pentru neuron adaptat implementării reţelelor neuronale mutistrat, cu învăţare online, care poate fi configurat să lucreze cu funcţii de activare de tip sigmoidal, respectiv tangentă hiperbolică. Pentru testarea performanţelor sale, neuronul propus a fost implementat pe un set de dispozitive FPGA. Rezultatele obţinute după implementare indică faptul că acesta necesită mai puţin de 1% din cantitatea totală de resurse logice ale dispozitivului FPGA, în timp ce frecvenţa maximă de lucru a acestuia poate fi crescută la valori mai mari decât 100 MHz.