POLLACK PERIODICA
An International Journal for Engineering and Information Sciences
DOI: 10.1556/Pollack.4.2009.3.13
Vol. 4, No. 3, pp. 143–154 (2009)
www.akademiai.com

EMBEDDED NEURAL CONTROLLERS BASED ON SPIKING NEURON MODELS

László BAKÓ, Sándor Tihamér BRASSAI

Department of Electrical Engineering, Faculty of Technical and Human Sciences
Tirgu-Mures, Sapientia - Hungarian University of Transylvania, Calea Sighisoarei 1C
Corunca, Romania, e-mail: lbako@ms.sapientia.ro

Received 12 January 2009; accepted 20 March 2009

Abstract: This paper demonstrates, that input patterns can be encoded in the synaptic weights by local Hebbian delay-learning of spiking neurons (SN), where, after learning, the firing time of an output neuron reflects the distance of the evaluated pattern to its learned input pattern thus realizing a kind of RBF behavior. Furthermore, the paper shows, that temporal spike-time coding and Hebbian learning is a viable means for unsupervised computation in a network of SNs, as the network is capable of clustering realistic data. Then, two versions - with and without embedded micro-controllers - of a SNN are implemented for the aforementioned task.

Keywords: Embedded systems, Hardware/software co-design, Spiking neural networks, FPGA

1. Introduction

The neuron models involved in Spiking Neural Networks (SNNs) are typically more complex than in conventional rate-coded artificial neural networks and the information passed between these neurons is expressed as temporally separated discrete events or spikes. SNNs and Pulse-Coded Neural Networks can generate behaviors and reproduce coding schemes closely analogous to biological neural systems. On the other hand, these types of neural models may present important advantages in terms of digital hardware implementation, due to the fact, that timing delay coding offers considerable resource utilization gains and it is more robust than the analog implementation [1], [2]. An interesting proposition related to the possible applications of networks of spiking

neurons has been published in the paper [1], where the authors showed, that a randomly connected network of spiking neurons can efficiently implement a temporal filter.

Learning rules, similar to those used in sigmoidal neural networks, can be devised to suit spiking neural networks, as well, for example a back-propagation-like rule presented in [2], named Spike-prop. There are also other interesting applications in the literature, that are proposing the use of spiking neural networks as a Bayesian interface, for instance in [3], [4] and [5].

The first implementation presented in this paper has been designed so that all the neurons and their components are instantiated as a separate module in the Xilinx Spartan-3 FPGA, circuit chosen as the hardware platform. This yielded a completely parallel hardware SNN architecture, with 24 input neurons, 72 synapses and 3 output neurons.

The second implementation partly sacrifices the fully parallel nature of the design, by embedding soft-core micro-controller modules (Xilinx PicoBlaze). The assembly written code running on these parts, replacing the network input computing logic (encoding the input values into delayed spikes - input neurons) and optionally the soma (cell body) modules of the spiking neurons. By doing so, the SNN's epoch computation time has increased slightly, but some 5% has been gained in terms of slice utilization and available BlockRAM modules. Furthermore, this yielded the possibility of enhancing the precision of the input variable spike coding as well as the clustering capability of the SNN. Implementing a three input network will also be possible.

2. Unsupervised classification of clusters using spiking neural networks

Inspired by the local receptive fields of natural neurons, the input values of the implemented SNNs have been encoded by overlapped and graded sensitivity profiles. This multiple encoding assures, that clusters will be classified flexibly. This approach is vital in unsupervised algorithms, since the scaling information is a-priori unknown.

Extending the network to multiple layers it can be demonstrated, that the sequential nature of pulsing neurons can be exploited to validate the correct classification of overlapping clusters. It is known, that in a multilayer Radial Bases Function (RBF) neural network, the neurons of the first layer are focused on cluster components. The design of this model ensures that the firing times of its output pulses depend on the synchronism and the occurring time of the synchrony of the input pulses. The synchronism in turns can be optimized by coordinating the relationship between the pattern of the input pulses and the synaptic delay pattern.

Most applications of pulsed and conventional neural networks are implemented using software simulator. Moreover, the computations in the aforementioned SNN are also quite complex for hardware implementation. This limits one of the most important advantages of neural networks, the massive parallelism. This paper presents the design of hardware circuits for the proposed neural network using field programmable gate arrays (FPGAs). As FPGAs are digital systems, the advantages of digital technique like robustness to noise and design flexibility are naturally effective.

2.1. Implementing a pulsed neural network with spike delays

The architecture is feed-forward, using leaky integrate-and-fire neurons. The connections of the network are composed of a set of k synapses, each with a w_{ij}^k weight and a d^k delay (*Fig. 1*). An input pulse from neuron I generates a set of post-synaptic weighted potentials (PSP, a function that models the impact of an input pulse on the target neuron as a temporal function). The magnitude of the synaptic weight determines the height of the PSP. In each time step, these PSP values are added to form a membrane potential at the target neuron. Given that this sum exceeds a predefined threshold θ, the J output neuron will generate an axonal spike [12], [13].

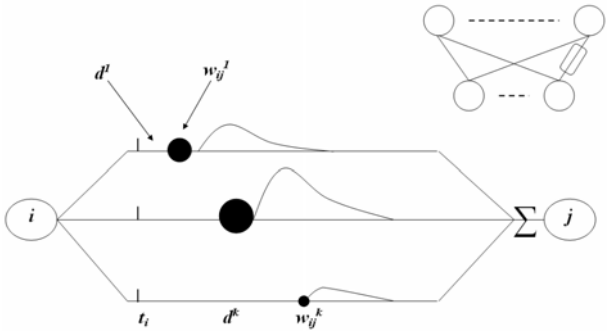

Fig. 1. The theoretical concept of the implemented neural network

In order to achieve unsupervised learning, the weights are tuned according to a temporal version of the Hebb algorithm. If a PSP precedes closely the arrival of an axonal activation pulse, then the weight is increased, because it is considered to be of high efficacy in increasing the membrane potential of the post-synaptic neuron. Other synapses, which receive input spikes at greater relative temporal distance from the axonal spike, will have their weights decreased, reflecting their reduced addition to the post-synaptic membrane potential. For a weight with a delay d^k from neuron I to neuron J, the a proper formula would be

$$\Delta w_{ij}^k = \eta L(\Delta t) = \eta(1-b)\exp\left[\frac{(\Delta t - c)^2}{\beta^2}\right] + b\,, \tag{1}$$

according to [2] (*Fig. 2*). The weights are protected against overflow and underflow events. The input values are encoded into delayed spikes emitted by the input neurons. Each input neuron is allowed to emit a single spike during the encoding time frame.

2.2. Encoding integer input values into spike delays

After studying [6], [7] and [8], several schemes, using multiple domain receptive fields have been investigated in order to extend the encoding precision and capacity, distributing an input variable through multiple input neurons. Distribution codes, where the input variables are encoded with integrated and overlapping functions, can often be found as efficient methods to represent real values. In these studies, the activation function of an input neuron is modeled as a receptive field that determines the activation rate. Translating this paradigm into relative activation moments is relatively simple: an optimally stimulated neuron will fire at time $t = 0$ [time step nr.], while an activation time stamp of up to $t = 15$ [time step nr.] is assigned to neurons with weaker stimulation (*Fig. 2*).

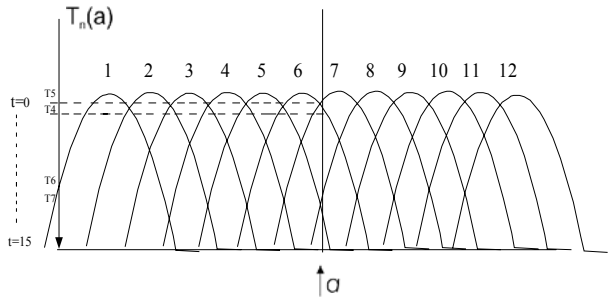

Fig. 2. The method of encoding the input variables into delayed spikes

In order to encode multidimensional data in the manner presented above, a choice has to be made regarding the dimension of the neuron's receptive fields. The least expansive way to do this, in terms of number of neurons needed, is to encode each input with 1-D, independent receptive fields. Since the focus of this paper is on multidimensional classification, this encoding is most convenient, because it is linear with regard to the number of needful neurons per dimension and it is, also, adaptable allowing the dimension encoding with higher precision, without excessive neural costs.

3. Implementing the input neurons - encoding the input variables into spike delays

Several versions of encoding algorithm have been experimented, considering the reconfigurable resources available in FPGA chips (Xilinx Spartan 3 XC3S1000, Spartan 3E XC3S1200E) of the utilized development boards. In order to make the best use of the reconfigurable logic blocks, specifically, to free up as many logic cells as possible for the implementation of spiking neurons, different approaches have been tested, to store the values of the RBF functions of the input neurons in BlockRAM modules. These modules are available in several configurations, with different widths for the address and data buses. The total of 54Kbytes available in these BlockRAMs is divided into 24 components. The designer's dilemma is to decide how to use these resources,

sacrificing - even if only partially - the pure parallel nature of the implementation or to exploit more BlockRAMs then it would actually be necessary to store the targeted data [14], [15], [16]. The number of input neurons as well as the size of the input space has also been varied, during experiments, in order to find the most suitable configuration.

As *Fig. 3* presents (generated by a C program), the functions of the receptive fields of the input neurons are not Gaussian, but triangular, to simplify the hardware implementation. The *x*-axis in *Fig. 3* holds the input values given to the network, values that are positive integer numbers between 0 and 192. Each value of this interval is distributed to a few input neurons (min. 2 - max. 4), depending on the number of receptive fields that it activates. The larger the value generated by the activated receptive fields (twelve input neurons used in this project) the sooner the respective input neuron will emit a pre-synaptic spike, meaning, that the temporal delay between the moment of input value arrival and spike emission will be accordingly shorter. Obviously, there are many possibilities to vary the overlapping areas between the receptive fields, thus modifying the number of input neurons that will activate for a given input value in order to encode that value. It is important to notice, that each input neuron will activate only for a reduced number of input values with a properly delayed spike. These values might be calculated in-circuit, but it would consume precious resources and computing power. Therefore, it has been decided to pre-compute these values using a C program and then to store them in the BlockRAM modules of the FPGA circuit. Using this program it is easy to vary the overlapping areas of the receptive fields as well as the number of input neurons used. On the other hand, the program generates the VHDL code for the BlockRAM initialization. The optimal utilization of the BlockRAM modules would have been achieved if the delay values of several input receptive fields had been stored in the same memory component. However, this would have lead to the impossibility of simultaneous access to all these memories. Therefore, a choice has been made to use more BlockRAMs, configured as 1024 words of 4 bits (RAMB_16_S4 Xilinx Spartan3 primitive) that act as a static, synchronous memory.

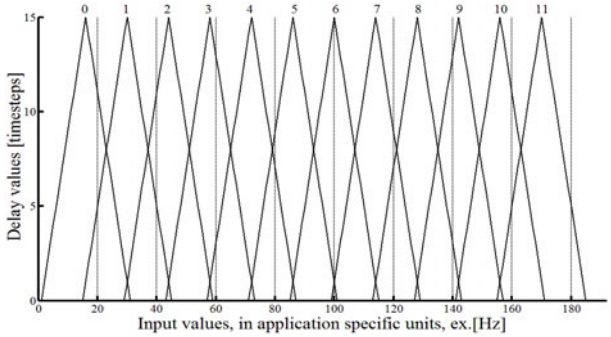

Fig. 3. Triangular receptive field functions used in the hardware implemented input neurons

This method has an important flaw, since it consumes all 24 BRAM modules available in the used Xilinx Spartan 3 XC3S1000 FPGA circuit, for the 12 input

neurons of each input of the neural network, with a memory utilization of about 3%. To solve this problem, a dual-port BRAM component has been used (RAMB16_S36_S36), that has a nine bits wide address bus and a 32 bit data bus, with a capacity of 512x12 bits. The dual-port architecture allows for two different memory locations to be accessed for reading simultaneously, hence by connecting in parallel two modules of this type, a 64-bit word can be accessed. This has been used, to read from these two memory modules, a 4-bit value for each of the 16 activation functions of the 16 input neurons. These values will determine the moment when the input neurons will emit an output spike, relative to the moment of appearance of the new input value to be encoded (steps 0 to 15 of each time frame). The values given as addresses to the memories (*Fig. 4*) are actually the input values of the SNN (offset incurred for port *B*).

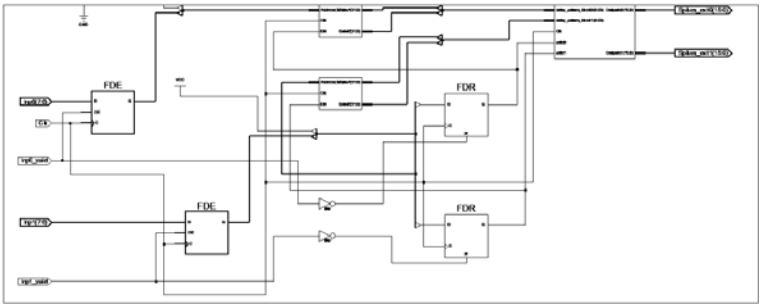

Fig. 4. The structure of the input coding unit

In order to facilitate the proper stimulation of the output neurons an alternative method has been devised, that guarantees, that more (three or four) input neurons will activate for a certain input values. Therefore, the values of the 16 triangular functions have been re-computed, increasing the overlapping sections. This would have increased the length of the time frame, change impossible to implement, since the BRAM's were already 100% filled with data. Scaling the new values (originally 0 to 35) to the given time frame size of 15, proved to be the easiest solution, that yielded the data in *Fig. 5*.

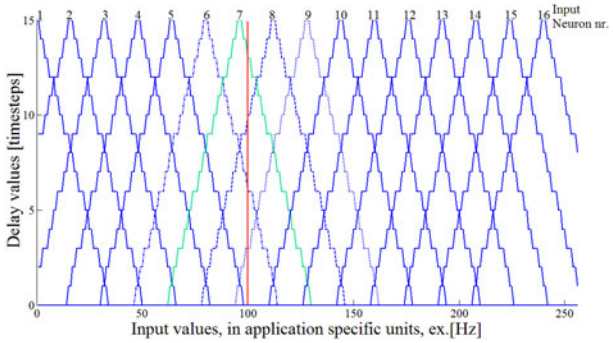

Fig. 5. Triangular functions of the encoding receptive fields, with scaled delay values for 16 pseudo-RBF-Spiking input neurons and an input space of 256x256 points (X axis)

4. The FPGA implemented pulsing neural network

The hybrid, pseudo-RBF-spiking neural network (psSNN) has been implemented with two inputs, describing an input space of 256x256=65536 points (*Fig. 6*). The aim of the implementation is to identify the points that form a cluster surrounding the trained focus-points. After learning, each of the three output neurons will tend to activate when the input values define a point in the input space that belongs to a cluster.

Fig. 7 presents the block diagram of the implemented psSNN.

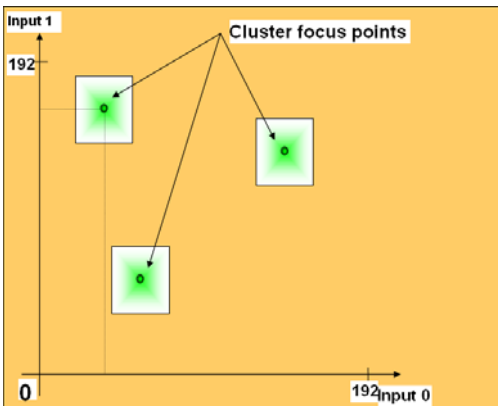

Fig. 6. The input space of the solved problem, with focus points and attached clusters

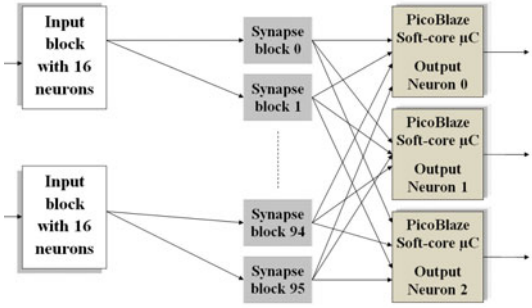

Fig. 7. Block structure of the implemented SNN

Each module of the network has been simulated either using Modelsim XE Starter or Xilinx ISE Simulators. In the following sections, the presentation of these modules will follow, detailing their role and function, as well as some simulation results.

4.1. The input neurons' block

This module is generated twice, one for each input variable (integer values on 8 bits). It contains the pair of BRAM's initialized with the pre-computed activation function values and a functional unit for each of the 16 activation functions.

The schematic in *Fig. 8* is a version of this input neurons block, used to test the proper operation, by studying the extra outputs inserted for debugging purposes. The functional module is a multi-state automaton that, extracts from the BlockRAMs the corresponding delay d^k values obtained from the stored receptive field function.

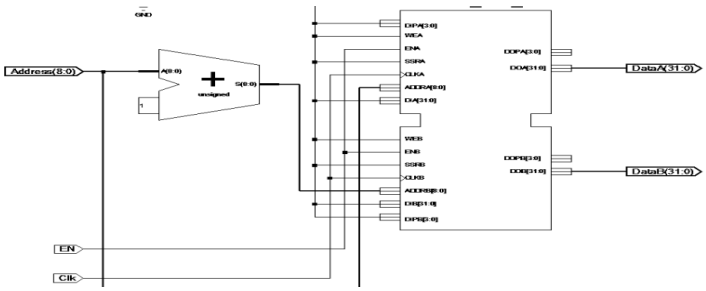

Fig. 8. Connection schematic of an input neuron

Using this value, a pre-synaptic spike is generated with the temporal delay d^k in parallel for each of the 32 input neurons of the two input blocks. *Fig. 9*, *Fig. 10*, *Fig. 11* and *Fig. 12* present the simulation test results for a sub-module of the input-processing element.

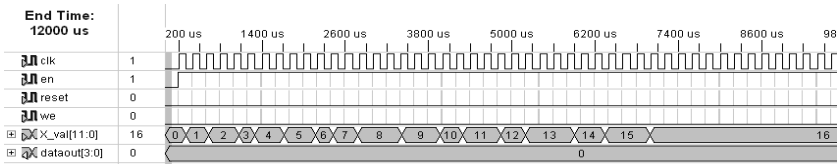

Fig. 9. Input stimulus for the BRAM module

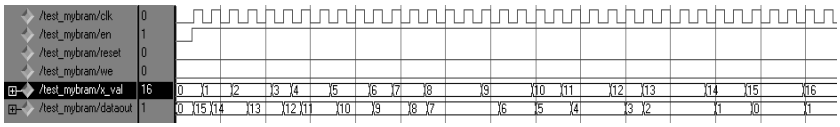

Fig. 10. Simulation result, with delay values (data-out) of for increasing input (xval)

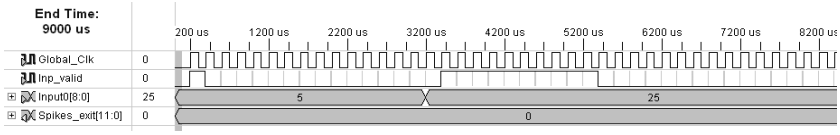

Fig. 11. Stimulus for testing the input encoding into delay values

4.2. Operation of the synapse module

This module is responsible for the amplification or attenuation of the input spikes, sending a weighted value to the neuron soma. This module also implements the

unsupervised learning algorithm, which is a novel, adapted version of the Hebb method, based on temporal delay rules, as described below. The synapses will test the time stamp of each incoming pre-synaptic pulse and will adjust the weight value according to a temporal version of the Hebb rules. Therefore, during a time frame of 16 clock cycles, for those pulses, that arrived with at most 5 cycles before the post-synaptic activation (of the corresponding output neuron, axonal pulse) the weight increases sharply. In the case of pulses, that have a delay between 5 and 10 clock cycles, the weight is increased moderately. For the rest of the delayed pulses, the synapse will be slightly weakened. When a pre-synaptic pulse arrives after the axonal one, weights are heavily decreased.

Fig. 12. Simulation result of the input encoding. See, that for ex. the input value 25 activates neurons 0 and 1, with delay values of 5 and 12 (clock cycles) while the other neurons will activate at the end of the time frame (generated by a dedicated module)

The inputs and outputs of this module are used to control the learning process and to load or store the weight values. In the project presented, there are a total of 96 synapses, 16 connecting each input neuron with each output neuron, storing the weights on 8 bits.

4.3. The output neuron module of the psSNN

The soma unit or output neuron is responsible for the summation of the arriving, weighted and temporally delayed, postsynaptic values. Each of the three-soma units of the implemented network receives 32 inputs of 8 bits each. By summing these values, and following the same time step pace then the synapses, the somas compute the membrane potential (MP) of the cell. If this MP exceeds a predefined threshold value, than the spiking neuron will fire (emit an axonal spike), and the MP will be reset to a special, hyper-polarization level, which is lower then the resting potential. If no incoming, postsynaptic values are received during a certain time step, then the MP will decrease linearly, thus implementing a kind of leaky integrate-and-fire neuron.

Implementing this module is one of the key issues of the project, because it presumes the summation of 32 variables on 8 bits, comparisons and register value adjustments, all performed in parallel. All these operations inherently lead to high FPGA hardware resource utilization. Setting as aim to implement a fully parallel architecture for the psSNN, using distinct reconfigurable elements of the FPGA for each task, thus performing concurrent computations, would yield in a very constrained size of the neural network, due to the limited number of reconfigurable blocks available. During the theoretical research preceding the implementation a partial serialization of the network has been projected using a Xilinx Virtex4 FX12 FPGA, that contains an embedded hard-core micro-controller, a PowerPC. This micro-controller would have

been used to compute serially the necessary data for several components of the SNN. Eventually, this version has been put on hold, due to the fact, that it would have needed a full re-design of the whole project. Nevertheless, a similar approach is much feasible and does not require a different hardware platform. Software simulations have concluded, that the partial serialization can be implemented using micro-controllers with much smaller footprint and computing capacity then the PowerPC, with the condition, that several of these controller to be implemented in the same design. Hence, an approach, that uses soft-core embedded micro-controller proved to be the best choice.

The most advanced soft-core embedded micro-controller for Xilinx FPGA's is MicroBlaze, delivered by Xilinx as a set of VHDL libraries and modules. However, the complexity of MicroBlaze is comparable to that of the PowerPC, so it is not a suitable part to use in the present design, since several instances of it would use too many resources. Hence, it was considered, that a very compact soft-core embedded micro-controller, the Xilinx PicoBlaze to be used, replacing the parallel implemented soma modules.

The following section presents the main characteristics of the PicoBlaze micro-controller. Then, a comparison between the fully parallel and the partially serialized implementations will be given, regarding the resource utilization, execution speed and the overall structures.

4.4. The Xilinx PicoBlaze (KCPSM3) soft-core micro-controller used to implement the soma modules

PicoBlaze is an 8-bit RISC micro-controller with a footprint of only 96 Spartan3 slices, which is about 1.25% of the total of 7680 slices of the Xilinx XC3S1000 FPGA used. The highest clock frequency it supports is 87MHz, that yields a respectable ~43.5 MIPS.

One can notice in *Fig. 13*, that the PicoBlaze has a 64-byte Scratchpad RAM and is able to interface with the surrounding logic implemented in the reconfigurable space of the FPGA through 256, 8-bit ports.

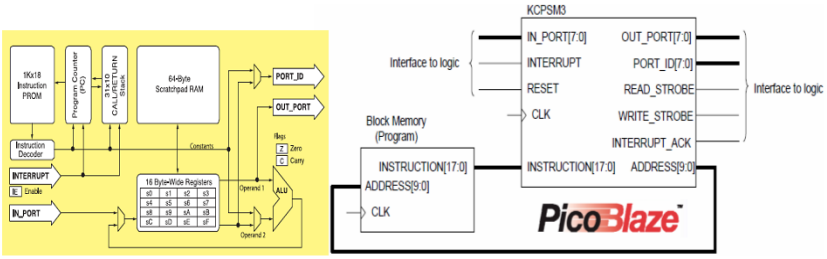

Fig. 13. Internal structure and block diagram of the Xilinx PicoBlaze soft-core micro-controller embedded in the SNN design

The programs are written for the PicoBlaze using the proprietary assembly instruction set, than compiled by the assembler delivered by the producer. This software

also generates the VHDL code necessary for the instantiation of the micro-controller in the user's design with the program pre-loaded into a BRAM configured as ROM.

By implementing in the PicoBlaze software the functionality of the soma modules, important reconfigurable resources have been freed up, allowing the extension of the psSNN (introducing more inputs). This will permit the implementation of more complex application of these novel types of NNs, such as the intelligent control of a robot arm or a mobile robot.

On the other hand, a side effect of this partial serialization is the reduction of the execution speed. This is a natural consequence, since the PicoBlazes need more time to compute sequentially the calculations of the soma algorithm. The negative effect of this issue has been significantly diminished by driving the PicoBlaze modules at higher clock frequency, than the rest of the components of the psSNN, using the DCMs (Digital Clock Managers) of the Spartan FPGA.

Table I summarizes and compares the reconfigurable hardware utilization of the two versions of the implemented pseudo-RBF-Spiking neural network.

Table I

Comparison of the resource utilization of the two methods of implementation of the psSNN, fully parallel and using PicoBlaze soft-core embedded micro-controllers

Implementation of the soma module	Slices		Flip-Flops		LUTs		Total gain
XC3S1000 Resources	Nr.	%	Nr.	%	Nr.	%	%
Total	7860	100	15360	100	15360	100	
Using PicoBlaze	130	1.65	76	0.49	171	1.11	1.09
Completely parallel	456	5.80	607	3.95	607	3.95	4.57
	Gain	3.51		7.99		3.55	5.01

In order to test the program running in the PicoBlaze units is fulfilling the required functionality, we used a software simulator (Mediatronix pBlaze IDE v3.6) and debugger. A test project has also been implemented to test the VHDL version of PicoBlaze, without the rest of the psSNN components.

5. Conclusions

The second implementation of a spiking neural network presented in this paper - with embedded soft-core micro-controllers - has proved to be more efficient in terms of FPGA hardware resource utilization, then other, fully parallel implementations presented here and compared to previous works, that implemented spike rate coded SNN. This justifies further research in this field, of SNN with partially serialized computation and RAM stored state variables for the components, which are also present in the recent specific literature.

On the other hand, in order to be able to implement a major application of these types of SNNs, with the complexity of an intelligent command and control of a robot

arm with four or more degrees of freedom or the control of mobile robots, a hardware platform with bigger capacity has to be used. The system used in the present paper is an XESS XSA1000 development board with a Xilinx Spartan 3 (XC3S1000) FPGA, that will be replaced by a larger board, the Digilent Nexys 2 with Xilinx Spartan 3E (XC3S1200E) FPGA.

References

[1] Maass W., Natschlager T., Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations, *Neural Computation*, Vol. 14, No. 11, 2002, pp. 2531–2560.

[2] Bohte S. M., Kok J. N., La Poutré H. Spike-prop: error-backpropagation in multi-layer networks of spiking neurons, *Neurocomputing*, Vol. 48, No. 1–4, 2002, pp. 17–37.

[3] Deneve S. Bayesian inference in spiking neurons, Ed. by L. K. Saul, Y. Weiss, L. Bottou, *Advances in Neural Information Processing Systems*, Vol. 17, MIT Press, Cambridge, MA, 2005, pp. 353–360.

[4] Rajesh P. N. R. Hierarchical bayesian inference in networks of spiking neurons, Ed. by L. K. Saul, Y. Weiss, L. Bottou, *Advances in Neural Information Processing Systems*, Vol. 17, MIT Press, Cambridge, MA, 2005, pp. 1113–1120.

[5] Richard S. Z., Quentin J. M. H., Natarajan R., Dayan P. Probabilistic computation in spiking populations, Ed. by L. K. Saul, Y. Weiss, L. Bottou, *Advances in Neural Information Processing Systems*, Vol. 17, MIT Press, Cambridge, MA, 2005, pp. 1609–1616.

[6] Snippe H. P. Parameter extraction from population codes: A critical assessment, *Neural Computation*, Vol. 8, No. 3, 1996, pp. 511−529.

[7] Marchesi M., Orlandi G., Piazza F., Uncini, A. Fast neural networks without multipliers, *IEEE Transactions on Neural Networks*, Vol. 4, 1993, pp. 53–62.

[8] Hikawa H. Frequency-based multiplayer neural network with on-chip learning and enhanced neuron characteristics, *IEEE Transactions on Neural Networks*, Vol. 10, 1999, pp. 545–553.

[9] Natschläger T., Ruf B. Spatial and temporal pattern analysis via spiking neurons, *Network: Comp. Neural Systems*, Vol. 9, No. 3, 1998, pp. 319–332.

[10] Bohte S. M., Kok J.N., La Poutre H. Unsupervised classification in a layered network of spiking neurons, *Proceedings of IJCNN'2000*, Vol. IV, 2000, pp. 279–285.

[11] Ed. by Omondi A. R., Rajapakse J.C., FPGA implementations of neural networks, Springer, 2006.

[12] Schrauwen B., Van Campenhout J. Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic, *ESANN'2006 Proceedings - European Symposium on Artificial Neural Networks,* Bruges (Belgium), 26-28 April 2006, pp. 623−628.

[13] Bako L., Brassai S. T. Spiking neural networks built into FPGAs: Fully parallel implementations, *WSEAS Transactions on Circuits and Systems*, Vol. 5, No. 3, 2006, pp. 346−353.

[14] Brassai S. T., Bako L. Hardware implementation of CMAC type neural network on FPGA for command surface approximation, *Acta Polytechnica Hungarica*, Vol. 4, No. 3, 2007, pp. 5−16.

[15] Mate K., Szabo I., Miniature digital telemetric bioelectric recording, *Pollack Periodica*, Vol. 1, No. 3, 2006, pp. 115–127.

[16] Dürr F. P., Mattiussi C. Neuroevolution: from architectures to learning, *Evolutionary Intelligence*, Vol. 1, 2008, pp. 47–62.