BRIEFINGS IN BIOINFORMATICS. VOL 11. NO 3. 348–363
Advance Access published on 6 January 2010
doi:10.1093/bib/bbp066

Real-time classification of datasets with hardware embedded neuromorphic neural networks

Laszlo Bako

Submitted: 3rd September 2009; Received (in revised form): 4th December 2009

Abstract

Neuromorphic artificial neural networks attempt to understand the essential computations that take place in the dense networks of interconnected neurons making up the central nervous systems in living creatures. This article demonstrates that artificial spiking neural networks—built to resemble the biological model—encoding information in the timing of single spikes, are capable of computing and learning clusters from realistic data. It shows how a spiking neural network based on spike-time coding can successfully perform unsupervised and supervised clustering on real-world data. A temporal encoding procedure of continuously valued data is developed, together with a hardware implementation oriented new learning rule set. Solutions that make use of embedded soft-core microcontrollers are investigated, to implement some of the most resource-consuming components of the artificial neural network. Details of the implementations are given, with benchmark application evaluation and test bench description. Measurement results are presented, showing real-time and adaptive data processing capabilities, comparing these to related findings in the specific literature.

Keywords: *spiking neuron models; embedded design; hardware implementation; clustering; FPGA*

INTRODUCTION

Research into artificial neural networks (ANNs) has seen the development of a variety of neuron models from the initial McCulloch and Pitts concept to the more biologically realistic spiking models [1]. Recent trends in computational intelligence have indicated a strong tendency towards forming a better understanding of biological systems and the details of neuronal signal processing [2, 3]. Research in this area is motivated by the desire to form a more comprehensive understanding of information processing in biological networks and to investigate how this understanding could be used to improve traditional information processing techniques. Spiking neurons (SNs) differ from conventional ANN models as information is transmitted by means of spikes rather than by firing rates. It is believed that this allows SNs to have richer dynamics as they can exploit the temporal domain to encode or decode data in the form of spike trains. However, this has demanded the development of new learning rules drawing again on inspiration from biology. For example, Hebbian learning has been identified as a closely biologically related learning rule and more recent research has reported a spike timing dependent variation of this rule called spike timing dependent plasticity (STDP) [4], which modulates the synaptic efficiency of synapses. Other learning rules, similar to those used in sigmoidal neural networks, can be devised to be applied in spiking neural networks, as well, e.g. a back-propagation-like rule presented in [5], named Spike-prop. There are also other interesting

Corresponding author. Laszlo Bako, Sapientia University, Electrical Engineering Department, Research Group on Biological signal processing and artificial intelligence, Str. Sos. Sighisoarei 1C, Tg-Mures/Corunca, Romania. Tel.: +40-365-403033; Fax: +40-265-206211; E-mail: lbako@ms.sapientia.ro

Laszlo Bako received the BS and MS degrees in electrical engineering in 2000 and 2001, respectively from the 'Petru Maior' University, Tirgu-Mures, Romania. He will present his PhD thesis on Hardware Implemented Neuromorphic Neural Networks in November 2009 at the 'Transilvania' University of Brasov, Romania. He is an Assistant Professor with the Electrical Engineering Department, of the Sapientia University, Tirgu-Mures, Romania. His research interests include embedded systems, reconfigurable computing, artificial intelligence, and real-time systems.

applications in the literature that are proposing the use of spiking neural networks as a Bayesian interface [6–8].

The main contributions this article brings are the real-time, on-chip learning capabilities, the even quicker response times after learning, the favorable hardware resource utilization and the superior performance in solving classification problems. The implementation of a spiking neural networks presented in this paper—with embedded soft-core microcontrollers—have also proved to be more efficient in terms of FPGA hardware resource utilization, than other, fully parallel implementations that used spike rate coded spiking neural networks (SNN).

Software simulation of network topologies and connection strategies provides a platform for the investigation of how arrays of spiking neurons can be used to solve computational tasks. These simulations face the problem of scalability, because biological systems are inherently parallel in their architecture whereas microcontrollers or PCs are based on sequential processing architecture. On the other hand, these simulations could be implemented on computers with multi-core or many central processing units (CPUs) but it would be a huge waste of computing power, since the capacity of the CPU cores is far greater than what is needed for real-time computation of a single artificial neuron.

Therefore, it is difficult to assess the efficiency of these models to solve complex problems [9]. When implemented on parallel hardware, NNs can take full advantage of their inherent parallelism allowing for specific units to be designed and added that boost the computing speed. Hence, these circuits will run orders of magnitude faster than software simulations, becoming appropriate for real-time applications while staying low-cost. Developing custom application specific integrated circuit (ASIC) devices for NNs, however is both time consuming and expensive. These devices are also inflexible, for instance the modification of the basic neuron model would require a new development cycle to be undertaken. Field Programmable Gate Arrays (FPGAs) are devices that permit the implementation of digital systems, providing an array of logic components that can be configured in a desired way by a configuration bit-stream. The device is reconfigurable, so a change to the system is easily achieved and an updated configuration bit-stream can be downloaded to the device.

This article aims to report on the issues arising from the author's experience in implementing spiking neural networks on reconfigurable hardware. This enables the identification of a number of challenges facing the area in terms of creating large-scale implementations of spiking neural networks on reconfigurable hardware, particularly that operate in real time, and yet demonstrate biological plausibility in terms of the adaptability of the architecture. The presented implementations of a SNN, partly sacrifice the fully parallel nature of the design, by embedding soft-core microcontroller modules (Xilinx PicoBlaze and MicroBlaze). These microcontrollers do not use dedicated hardware resources of the FPGA platform. Instead, the vendor delivers these as a source code package written in a hardware description language (HDL). Therefore, soft-core microcontrollers consume the same reconfigurable space of a FPGA circuit as any related logic the designer might develop. The assembly or C language written code running on these parts replaces the cell body computation logic. By doing so, the SNN's epoch computation time increases slightly, but important gains can be found in terms of slice utilization and available embedded memory modules, also known as BlockRAM modules. 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 network with more inputs will also be possible.

The classification of datasets using spiking neural networks presented in this paper is 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. An approach like this 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. This model is designed so 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 simulators. Moreover, the computations in the aforementioned SNNs are also quite complex for hardware implementation. This limits one of the most important advantages of neural networks, the massive parallelism. In this article, the aim is to design hardware circuits for the proposed neural networks using field programmable gate arrays (FPGAs). As FPGA circuits are digital systems, the advantages of digital technique like robustness to noise and design flexibility is naturally effective.

IMPLEMENTING A PULSED NEURAL NETWORK WITH SPIKE DELAYS

The architecture of the SNN 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 (Figure 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. The values given as inputs to the network can be encoded into the synaptic values through delayed learning [10–12]. After the learning phase, the temporal moment of the activation of the output neurons will reflect the distance between the sample given and sample that has been learned, thus implementing a RBF-like neuron behavior.

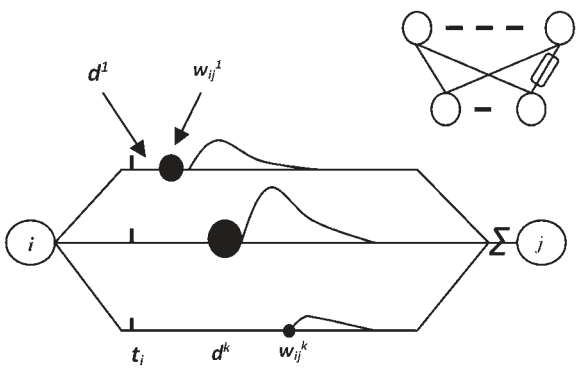

Figure 1: Theoretical concept of the implemented neural network.

The implemented spiking neural model

From a formal point of view, a neuron j, with a set of Γ_j pre-synaptic neurons receives a set of spikes with activation moments t_i. The dynamism of the internal state variable $x_j(t)$ (1) is determined by the arriving spikes, with the influence given by the spike response function $\varepsilon(t)$, weighted with the synaptic efficiency w_{ij}.

$$x_j(t) = \sum_{i \in \Gamma_j} w_{ij} \varepsilon(t - t_i) \tag{1}$$

The spike response function given in (2) models efficiently the un-weighted postsynaptic potential (PSP) for an input spike which affects a single neuron. The amplitude of PSPs to obtain the effective post-synaptic potentials are modulated with post-synaptic weights w_{ij}. A simplified version of the spike response function defined in (3) was used in experiments.

In the implemented network, an individual connection consists of a fixed number of synaptic terminals, where all the terminals constitute a sub-connection associated with a delay and a different weight.

The synaptic terminal delay d^k at synaptic terminal k is defined as the time difference between the pre-synaptic neuron activation time and the time when the post synaptic potential starts increasing.

A pre-synaptic spike at synaptic terminal k is described like a PSP with standard amplitude and a delay d^k. The un-weighted contribution of a single terminal to the value of the state variable is presented in the formula:

$$y_i^k(t) = \varepsilon\left(t - t_i - d^k\right) \tag{2}$$

with a spike response function form like a PSP, with $\varepsilon(t) = 0$ for $t < 0$.

Time t_i represents the activation time for the pre-synaptic neuron i, and d^k is the delay associated with synaptic terminal k. The response function, for a single spike which describes a standard PSP has the form:

$$\varepsilon(t) = \frac{t}{\tau} e^{1 - \frac{t}{\tau}} \tag{3}$$

modeling a simple α function for $t > 0$, and τ modeling the membrane potential delay time constant which determines the rising time and the delay for

a PSP. Expanding (1) to include multiple synapses in a connection, and inserting (2), the state variable x_j of neuron j, which acquires spikes from all the neurons I, can be described like a weighted sum of pre-synaptic contributions:

$$x_j(t) = \sum_{i \in \Gamma_j} \sum_{k=1}^{m} w_{ij}^k \gamma_i^k(t) \quad (4)$$

where w_{ij}^k represents the synaptic weight of terminal k. The activation time t_j for a neuron j is determined by the first time when the state variable exceeds the ϑ: $x_j(t \geq \vartheta)$ threshold. As a result, the activation time t_j is a nonlinear function of the state variable x_j: $t_j = t_j(x_j)$.

A novel hybrid learning rule is introduced, that mixes unsupervised (Hebb) and supervised algorithms. The weights are tuned according to a temporal version of the Hebb algorithm with regard to the desired output of the network, as described in the following sections. 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 a 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, a proper formula would be:

$$\Delta w_{ij}^k = \eta L(\Delta t) = \eta(1-b)\exp\left[\frac{(\Delta t - c)^2}{\beta^2}\right] + b, \quad (5)$$

according to [7], but this formula is unrealistic in terms of hardware implementations, hence a simpler method is needed. 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.

After studying [13–15], 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. A distribution code, where the input variables are encoded with integrated and overlapping functions, can be found as an often studied method 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 $t = 0$, while an activation timestamp of up to $t = 15$ is assigned to neurons with weaker stimulation (Figure 3).

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 expensive way to do this, in terms of number of neurons needed, is to encode each input with 1-D, independent receptive fields.

Since the interest is in multidimensional classification, this encoding is the 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.

In the following sections two benchmark test implementations will be presented aiming to assess the computational performance of this method, and proving that it could be suitable as an analytical tool in genetics or molecular biology.

BENCHMARK TEST IMPLEMENTATION—CLASSIFICATION OF THE FISHER IRIS DATASET

The first benchmark implementation is the classic test that proposes the classification of Fisher's IRIS dataset [16], solved using hardware implemented SNN. This test is often used even today, being a reference for many applications. The dataset contains three classes of fifty samples each, describing three types of Iris flowers by enumerating four attributes per sample.

The spiking neural network implemented on the FPGA circuit, that solves this benchmark test has four inputs and three outputs, as Figure 2 shows. In order to classify correctly the elements of the IRIS dataset the values of the attributes—presented as inputs to the network—were scaled (original values range from 0.1 to 7.9, these were scaled between 1 and 80) so that an integer representation to be possible in the FPGA circuit.

The SNN was trained to activate one of its output neurons only when the sample presented at the inputs belongs to the class associated with the respective output neuron.

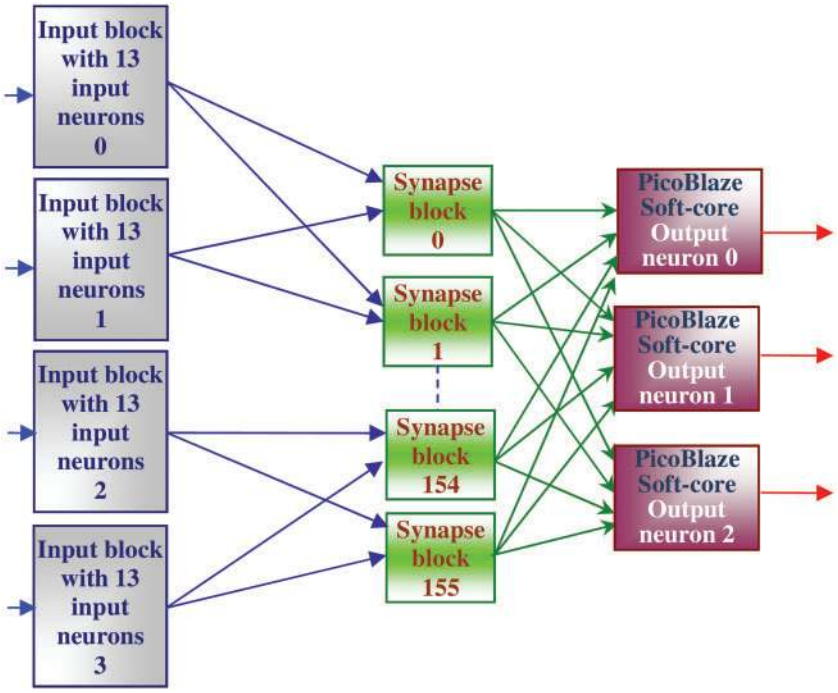

Figure 2: Block diagram of the implemented SNN, for the IRIS dataset classification.

Implementing the input neurons—encoding the input variables into spike delays

Having a hardware implementation as aim, a choice has been made to use a spike-time encoding scheme of the input variables instead of spike-rate coding. This simplified the circuit needed for this encoding, while using the inherent sequential nature of these digital systems to encode the information.

Only a small number of the synaptic weights are represented here, in order to show properly the effect of the learning mechanism. Note that some of the weight values do not change at all. This happens due to the fact that the input neuron, to which the respective synapse is connected to, did not fire for the given input values.

Several versions of the encoding algorithm (of the input values into delayed spikes) have been experimented, considering the reconfigurable resources available in the FPGA chip (Xilinx Spartan3E XC3S1200E) of the utilized development board. 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 functions of the input neurons in memory modules embedded into the reconfigurable FPGA circuit (BlockRAM) modules.

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 than it would actually be necessary to store the targeted data.

The input space defined by the IRIS attribute values could be covered smoothly by 13 input neurons with triangular activation functions, displayed in Figure 3.

It is easy to notice, that the vertical bar in this figure, representing the value 35 given as example input value intersects four of the thirteen triangular functions (numbers 5, 6, 7 and 8), thus activating the corresponding input neurons. Each of these input neurons will fire (emit an output spike) a number of cycles after it has received the input value, depending on the position of the intersection point with its activation function. The inactive neurons will not fire for this input 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 Matlab 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

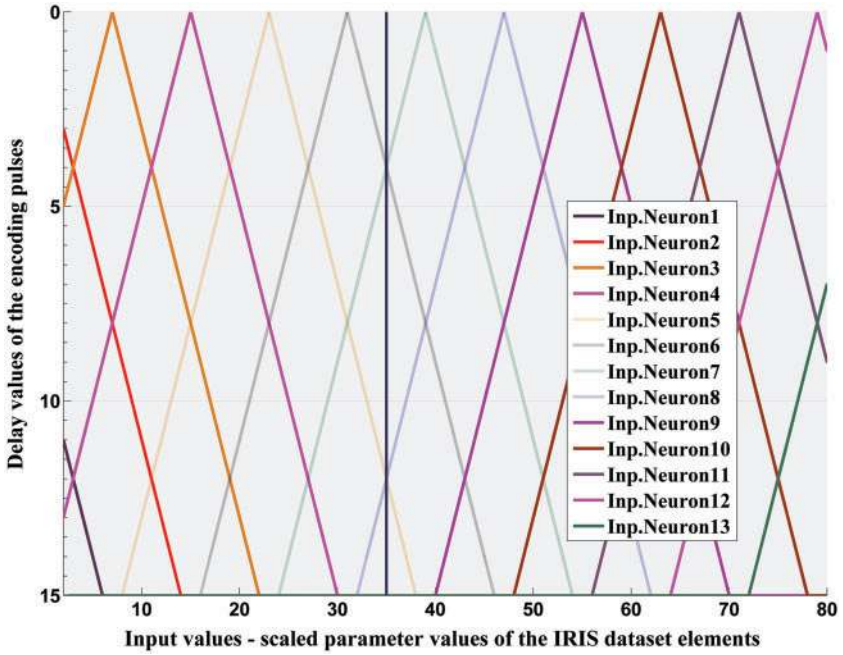

Figure 3: Triangular functions of the receptive fields, with scaled delay values (IRIS). As this figure presents, the functions of the receptive fields (generated by a Matlab program) of the input neurons are not Gaussians but triangular, to simplify the hardware implementation. The *x*-axis of Figure 3 holds the input values given to the network, values that are positive integer numbers between 1 and 80 (scaled values of the IRIS dataset element properties). Each value of this interval is distributed to four input neurons, activating as many receptive fields.

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.

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 0–15 of each timeframe). The values given as addresses to the memories are actually the input values of the SNN.

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 novel, supervised learning algorithm, which is an adapted version of the Hebb method, based on temporal delay rules, as described below.

Since it is easier to implement a rule based learning algorithm in hardware than one based on mathematical formulas, this has been an obvious choice. On the other hand, due to the need for a supervised method, a new rule-set has been devised, that can be considered as a new, hardware implementation-friendly learning rule for SNNs.

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 but will also take into account if the target neuron was supposed to fire for the actual input value pair or not. Therefore, as Table 1 shows, during a time-frame of 16 clock cycles, for those pulses, that arrived with at most five cycles before the post-synaptic activation (of the corresponding output neuron, axonal pulse) and if the corresponding neuron should spike (in order to correlate its activation with the presence of an input pair of the relating cluster) the weight increases sharply. In the case of pulses, that have a delay between 5 and 10 clock cycles and similar conditions, the weight is increased moderately. For those delayed postsynaptic pulses, that arrive after the axon has spiked the synapse will be weakened proportionally, but only if the axonal spike shouldn't be there for the actual input pair.

The inputs and outputs of this module are used to control the learning process and to load or store

Table 1: The implemented novel learning rule

Synaptic input present	Axon value -SN output	Time-step value rules	Target output value	Adapt weights	Compute somas
0	0	–	0	–	–
0	0	–	1	–	$\sum$
0	1	–	0	–	–
0	1	–	1	–	$\sum$
1	0	–	0	–	–
1	0	PreTS ‹‹ PostTS	1	↑↑	$\sum\sum$
1	0	PreTS ‹ PostTS	1	↑	$\sum$
1	1	PreTS › PostTS	0	↓	–
1	1	PreTS ›› PostTS	0	↓↓	–
1	1	–	1	–	$\sum$

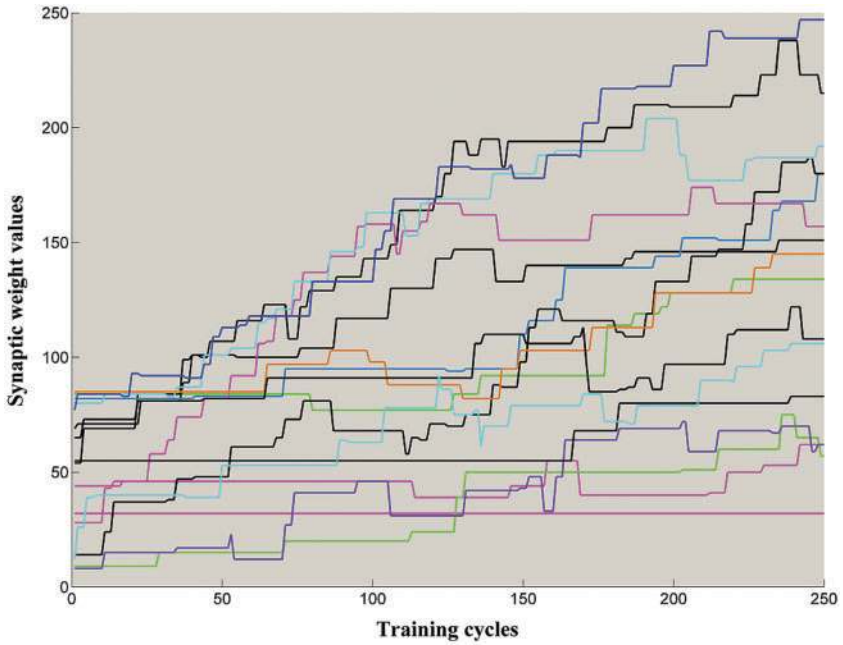

Figure 4: Changing weight values during learning.

the weight values. These randomly generated weight values (represented on 8 bits) of the synapses will be adapted according to the rules of the learning mechanism (Figure 4), as the inputs are presented to the network by a Control Unit that reads them from a pre-loaded BlockRAM memory.

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 output units of the implemented network receives a number (equal to the number of synapses in the project, that is 156, calculated by multiplying the nr. of inputs × nr. of input neurons for one input × number of output neurons - 4 × 13 × 3) of inputs represented on 8 bits each. By summing these values, and following the same time-step pace than the synapses, the somas computes the membrane potential (MP) of the cell. If this MP exceeds a predefined threshold value (THP), 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 than 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,

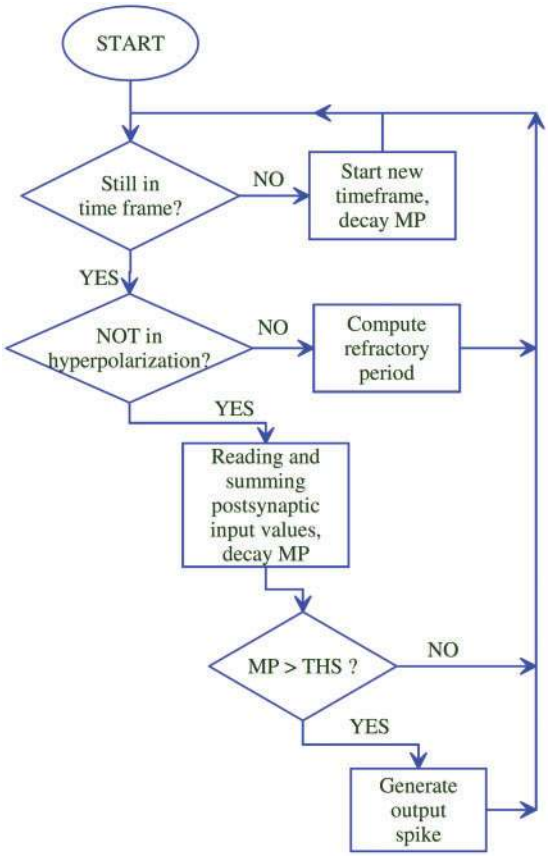

Figure 5: Flowchart of the PicoBlaze assembly code that implements the soma algorithm.

because it presumes the summation of more than 156 variables on 8 bits (flowchart of the implemented PicoBlaze assembly code can be seen in Figure 5), 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 SNN, 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.

The embedded Xilinx PicoBlaze soft-core microcontroller used to implement the soma modules, also known as KCPSM3, is an 8-bit RISC microcontroller with a small footprint of only 96 Spartan3 slices, which is about 1,10% of the total of 8672 slices of the Xilinx XC3S1200E FPGA used. The highest clock frequency it supports is 87MHz that yields a respectable ~43.5MIPS. By implementing in the PicoBlaze software the functionality of the soma modules, important reconfigurable resources have been freed up, maintaining real-time performance.

A very important and difficult to solve issue in this project was the timing of the different algorithms in the design running on different circuits (dedicated hardware resources or embedded microcontrollers). Therefore, all the synchronization is commanded by the three PicoBlazes implementing the somas, as these modules generate the clock signals for the other parts. Besides doing so, the soma algorithm is also executed. In order to be able to follow the evolution of the internal variable of the SNN (MPs and weights) a monitoring unit has been designed, featuring a separate PicoBlaze, which periodically reads these values and sends them to a connected PC.

Test bench system description

The entire neural circuit is controlled by a fourth embedded soft-core PicoBlaze microcontroller, shown as Monitoring Unit in Figure 8. This module commands the step-by step execution of the training process.

The flowchart of the assembly program running in this microcontroller is presented in Figure 10, while the phases and steps of the neural network operation are summarized in Figure 9.

The IRIS dataset has been split into a training set containing 80% of the samples and a test set with the rest of 20% of the elements.

The first loop in Figure 10 (epochs) goes randomly through all these input values of the training set, enabling the input generation module (Control Unit) to read these values from the attached BlockRAM memory. Then, the Input encoding block is enabled, and the input blocks containing the input neurons will read the corresponding delay values from their embedded BlockRAM memories. Afterwards, the encoding sequence (of 16 cycle timeframe) of these values into delayed spikes will be started with the proper command signals activated by the Control Unit PicoBlaze.

Next, the synapse modules are activated. At this point the somas have the first set of inputs at their ports, so their implementing PicoBlazes are activated to compute one time step. At the proper moments (Figure 9) the SNN's most important parameters (weights, MPs, Axon values) are read into the Monitoring Unit's controller, then sent via a

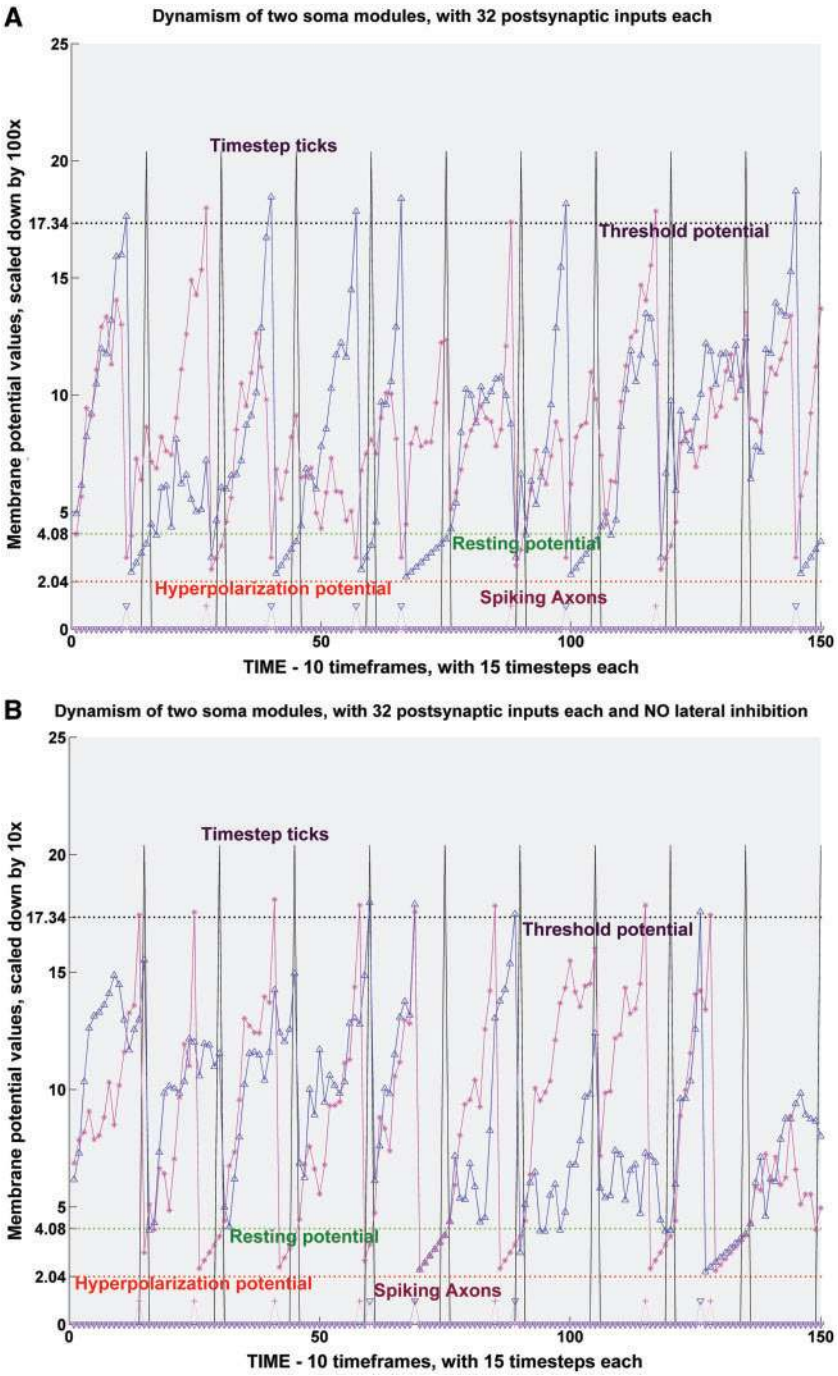

Figure 6: Soma dynamism with (**A**) and without (**B**) lateral inhibition. This figure presents two sets of these MP data, during the training of the SNN. The somas deliver laterally connected inhibitory signals to the other somas, in order to prevent them from spiking together, hence differentiating the individual components to be learned. If these inhibitory signals are removed, the somas will behave differently, as the figure on the right shows. Note that when one of the two somas is firing, the MP of the other neuron is reset to the resting value. After activation a neuron goes into hyper-polarization and according to the used model this neuron should not emit another spike so the MP will rise slowly, only to reach the resting value by the end of the current time-step.

UART module to a PC, for later analysis and visualization (Figures 6 and 7).

One of the great assets of the present implementation is that it performs in real-time, despite the fact, that the involved computation is partially serialized due to the use of embedded microcontrollers. These controllers operate at a clock frequency of 50MHz, executing an instruction in two clock cycles.

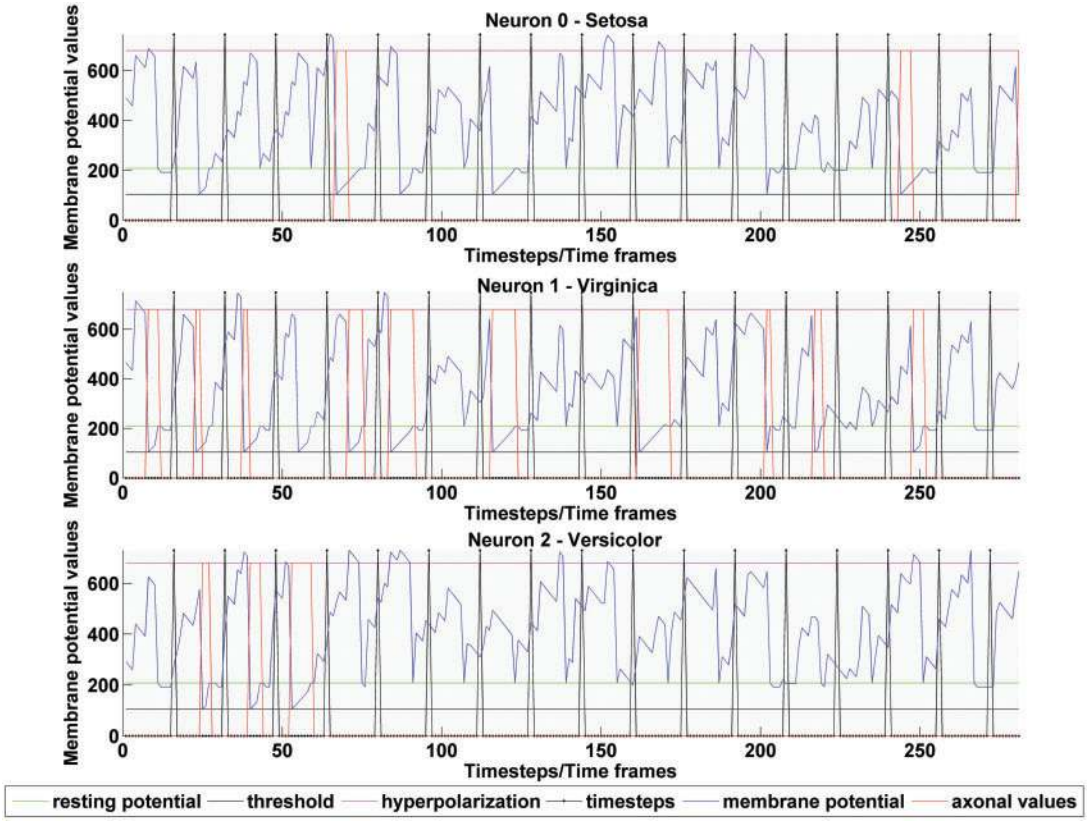

Figure 7: Measurement results on all the output neurons of the FPGA implemented SNN. The three subplots in this figure show the variation of the membrane potential in the output neurons. The specific parameters of the soma model are also plotted, to aid the readability of the figure. Also, the axonal values can be noted, showing when the particular output neurons identified the input pattern of the attributes as a sample of the IRIS dataset corresponding to one of the classes.

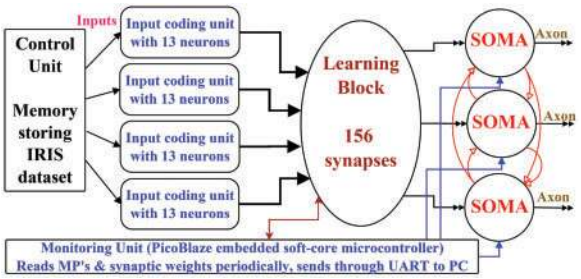

Figure 8: Block diagram of the test bench system.

The soma algorithm is implemented in a code of about 200 instructions, while all the other components of the SNN are implemented as modules operating in parallel. This yields a computation time/epoch of about 64 μs, with a learning time of 16ms. Therefore it can be stated, that this approach in implementing neuromorphic neural networks has great potential even in real-time applications.

BENCHMARK TEST IMPLEMENTATION—CLASSIFICATION OF THE WISCONSIN BREAST CANCER DATASET

The second benchmark implementation is the test that proposes the classification of Wisconsin Breast Cancer Dataset IRIS dataset (WBCD), solved using hardware implemented SNN. The dataset contains two classes (malign and benign) of features sets (computed from a digitized image of a fine needle aspirate (FNA)) of a breast mass. These nine features or attributes of each of the 683 elements of the dataset describe characteristics of the cell nuclei present in the image [17].

Since this is a more complex classification task, than the one previously presented, in this case the greatest computational load of the project has been assigned to be carried out by an embedded 32 bit soft-core processor, the Xilinx MicroBlaze.

Clock Cycle / Phase	a		b	c	d	e
Clock cycle nr.	Training Control	Inputs valid	Input block Clock	Synapses Clock	Somas (pBlaze) Clock	Send to UART
1	Rising edge	NO	no edge	no edge	HOLD	
2	Rising edge	NO	no edge	no edge	HOLD	
3	Rising edge	YES	no edge	no edge	HOLD	
4	Rising edge	YES	no edge	no edge	HOLD	input values
5	no edge	YES	Rising edge	no edge	HOLD	
6	no edge	YES	Rising edge	Rising edge	HOLD	weights
7	no edge	YES	no edge	no edge	Running	MPs
if TS_nr<16 goto 5				if TS_nr=15 then -->		weights, Axon values
if Epoch<129 goto 1						

Figure 9: Steps of the SNN computation.

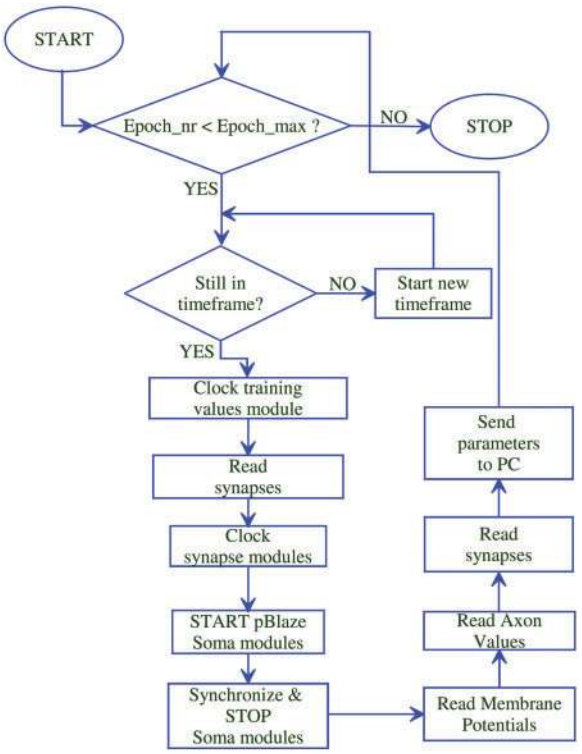

Figure 10: Flowchart of the PicoBlaze program of the Control and Monitoring Unit.

The MicroBlaze embedded soft core is a reduced instruction set computer, optimized for Xilinx FPGAs, implemented with Harvard architecture. It means that it has separate bus interface units for data and instruction access.

The input encoding unit (IEU) (into delayed spikes) has been designed as a separate peripheral device—in the remaining reconfigurable FPGA fabric –, and has been connected to the MicroBlaze on the Processor Local Bus (PLB).

The task of the input encoding peripheral is to convert those nine attributes of the elements in the WBCD, that are used as inputs to the hardware SNN, into spikes emitted by the activated input neurons at different time-steps of a given time-frame (similarly to the IRIS encoding, see Figure 11). These pulses (emitted by 4 input neurons out of 16 per input, a total of $4 \times 9 = 36$ in a time-frame of 15 cycles) are then transmitted to the MicroBlaze processor that performs the rest of the SNN computation.

The WBCD element attributes are values represented with one decimal place (range 0–10), hence a pre-scaling operation (to the range 0–100) was needed to ease the digital hardware storage (in embedded BlockRAMs).

The code running in MicroBlaze will extract an element of the WBCD from the BlockRAM storage by sending a proper command to the IEU as a 64 bit value containing all nine attributes (represented on 7 bits) and the class identifier as the 64th bit (0-benign, 1-malign).

The synapses (with the learning rules) and all the neuron bodies or somas (Figure 12) of the SNN (six in the hidden layer and two in the output layer) are implemented as functions of the

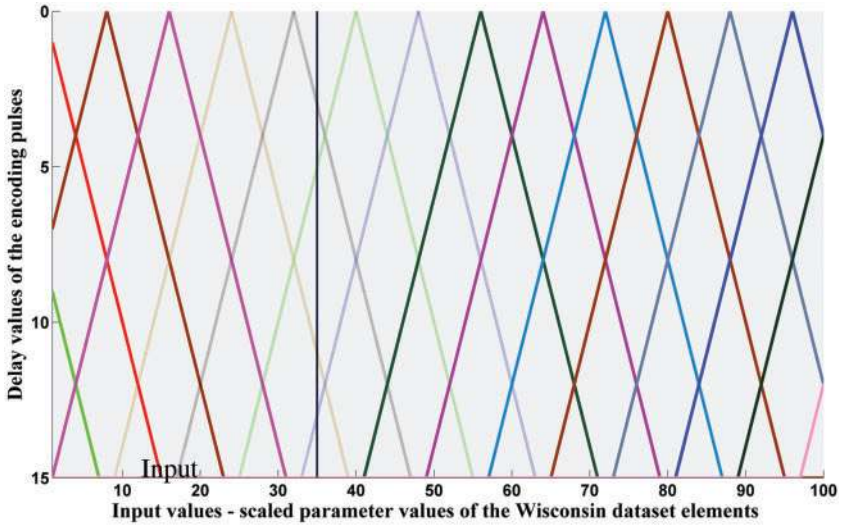

Figure 11: Triangular functions of the receptive fields, with scaled delay values (WBCD).

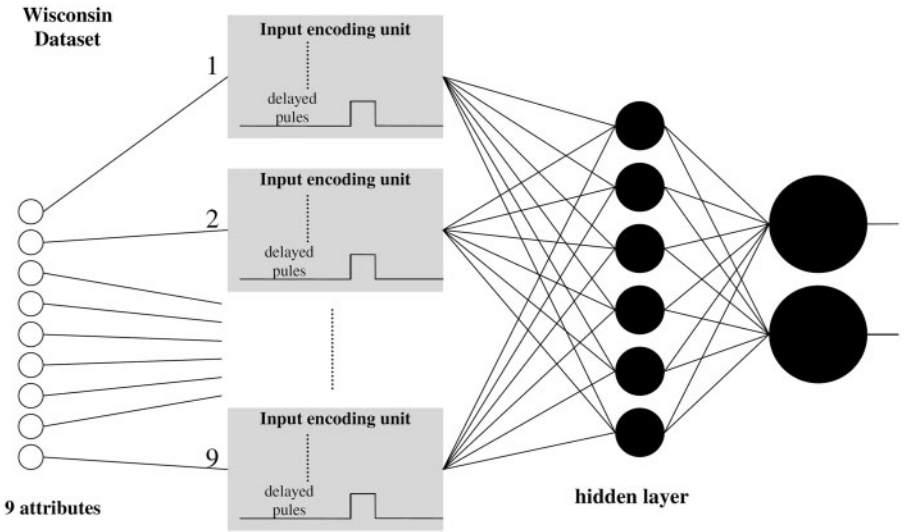

Figure 12: Block diagram of the implemented SNN, for the WBCD classification. The SNN contains 9 input encoding units with 16 input neurons each that turn the input values into time-delayed spike patterns. The network is fully connected, with the six hidden layer neurons connecting to the input neurons through 9*16*6 = 864 synapses. Further 12 synapses connect the hidden layer neurons to the two output neurons.

software executed by the embedded MicroBlaze processor. Since the SNN is fully connected the number of synapses—connecting the input neuron to the hidden neurons and these to the output neurons, respectively—is much larger than in the IRIS application. This number yields to 876, if we multiply the number of aforementioned elements (9 inputs × 16 encoding input neurons × 6 hidden neurons = 864) plus the 12 synapses connecting the hidden layer neurons to the output somas. This also results in a 4-fold increase in memory capacity to store the synapse variables (weights). Inevitably, the available BlockRAM capacity of the used FPGA circuit (Xilinx Spartan3E XC3S1200E) has been exceeded. Therefore, the external DDR RAM module of the development board had to be used to store the data segment of MicroBlaze software.

The role of the hidden layer neurons in this application is feature extraction form the 9D space defined by the inputs of the SNN, the WBCD attributes, leading to better classification results.

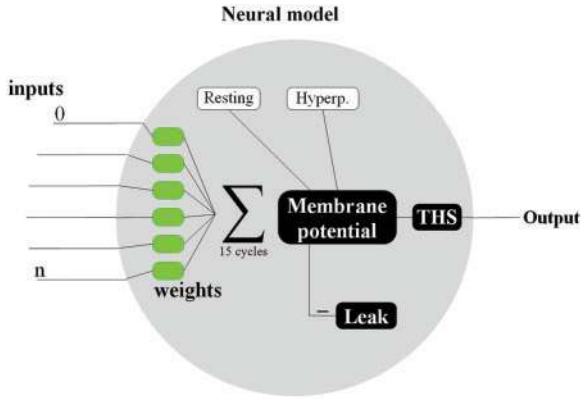

Figure 13: Model of the neurons used in the WBCD application. The soma accumulates the weighed post-synaptic values from the input encoding neuron assemblies into a membrane potential parameter. If this value exceeds the predefined thershold potential (THS), then the neuron will activate, emitting an axonal spike, or action potential. After activation, the neuron will enter a state named refractoy period, where further activations are inhibited due to the membrane potential being reset to a hyper-polarization value, well below the resting potential. If no inputs are received during a 15 cycle time-frame, the membrane potential's only change will be the loss of a charge equal to the leak value that eventually would lower it to the resting value.

The implemented learning algorithm

The neurons of the hidden layer are built on the same model as the one in the output layer, hence their operation is identical, both types being implemented as functions written in C language, to be executed by the MicroBlaze processor. Two groups of three were formed in the hidden layer, alternating their prescribed activation value according to the class of the current element presented at the inputs, showed in the WBCD.

Obviously, the WBCD has been split into a training set, containing some 65% of the original dataset's elements, the rest forming the test set.

The neurons of the output layer (Figure 13) have been trained to activate when at least half of the hidden neurons are active. The synaptic weight of all neurons will be adjusted with a value that depends on the number of the time-step in which the corresponding post-synaptic value has been received.

Hence, for post-synaptic values received in the first time-step—which means, that one of the input neurons was activated with the maximum value of its sensitivity profile—the connected synapse has to be strengthened firmly (it will be increased with 0.5% of the maximum value of the weight, for instance 60,

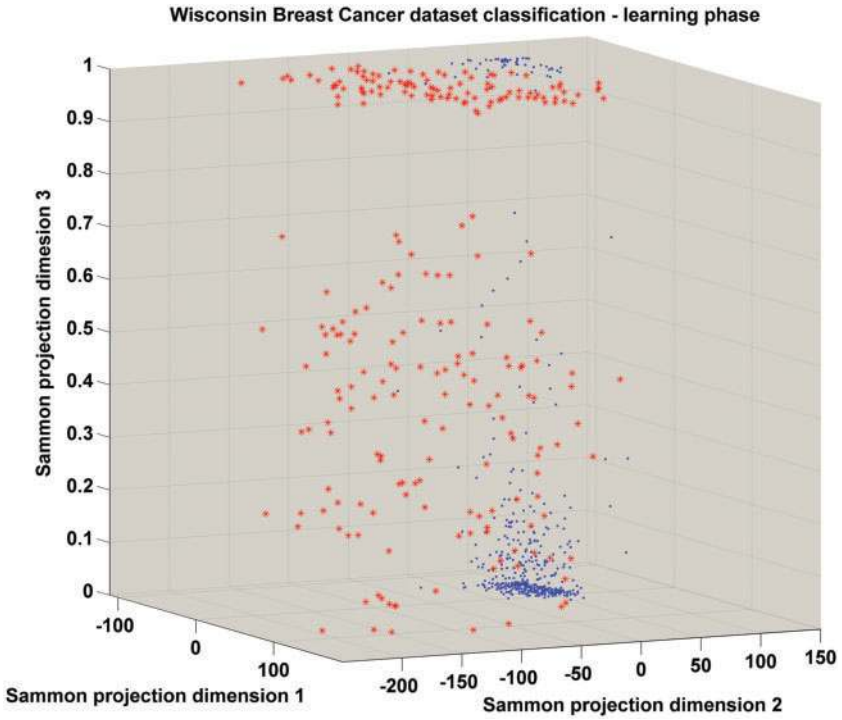

Figure 14: Sammon projection plot of an intermediary learning stage of the SNN implementing the WBCD classification. The smaller dots (blue) and the larger ones (red) denote elements classified to the two classes of the WBCD. The Z axis (Sammon dimension 3 or hight) can be interpreted as the error percent (scaled between 0–1) of the classification accuracy. This means, that elements plotted on the base of the cube are already 100% trained, the error increasing towards the top.

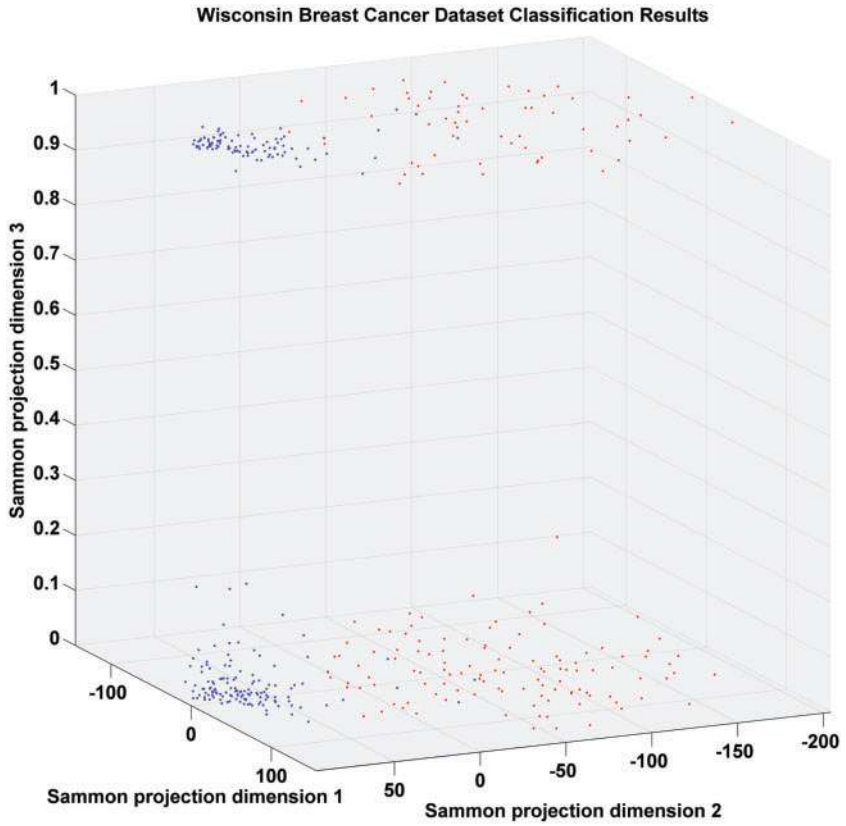

Figure 15: Sammon projection plot at the end of the learning stage of the SNN implementing the WBCD classification. Darker and lighter dots denote elements classified to the two classes of the WBCD. The Z-axis (Sammon dimension 3 or hight) can be interpreted as the error percent (scaled between 0 and 1) of the classification accuracy. This means, that elements plotted on the base of the cube are already 100% trained, those that stayed at the top have not been properly trained, but might deliver correct classification on the test set with lower probability.

weights ranging 0–1200). When a synapse will emit a post-synaptic value in a later time-step, the axonal value is also active, so is the prescribed, the weight will be increased with the time-step number's value, according to the following formula: (weights were initialized with random values limited to $w_j^{ik}{}_{\max} * 0.3$):

$$\Delta w_j^{ik} = \frac{w_j^{ik}{}_{\max}}{m} \tag{6}$$

where $m =$ [1/14] is the time-step number. If a neuron activates when the prescribed value does not require it to, the weights that contributed to this activation are weakened.

Sammon's projection [18] (used to plot Figures 14 and 15) is a nonlinear projection method to map a high dimensional space onto a space of lower dimensionality. The algorithm maps the original space onto the projection space in such a way that the inter object distances in the original space are being preserved in the projected space.

CONCLUSIONS

The SNN implementing the IRIS dataset classification has shown an error rate of about 18% after 300 training cycles. On the other hand, the hardware SNN that performs the classification of the WBCD did not manage to classify correctly the presented elements in ~8% of the cases, after 150 training cycles. It is very important to mention, that after the on-chip training has finished, in the testing phase both applications performed in real-time (IRIS: sub-millisecond, WBCD 5–10 ms), despite the fact that both use embedded soft-core processors, that partially serialize the computations.

The plot in Figure 16 compares the performance of the presented IRIS benchmark test's hardware

SNN implementation, with other known methods, that are usually implemented on general purpose single or multi-core CPUs. A similar chart, in Figure 17, displays the comparison of the WBCD classification solving hardware SNN's accuracy with software implemented methods.

Although, the accuracy of the implemented systems, presented by these charts, does not excel amongst the other methods, the main contributions they bring are the real-time, on-chip learning capabilities, the even quicker response times after learning and the favorable hardware resource utilization. The yielded computation time/epoch of about 64 μs, with a learning time of 16ms of the IRIS implementation, for instance, outperforms the results presented in [19], where the authors report, that their FPGA implemented, genetically evolved ANN system takes approximately 65ms to complete the clustering of the Fisher's Iris data.

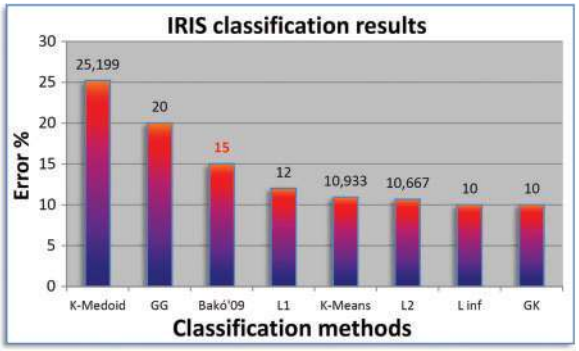

Figure 16: Comparative plot of the implemented SNN's performance with other methods (IRIS).

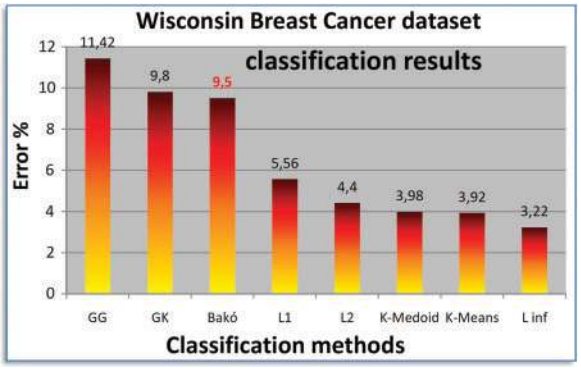

Figure 17: Comparative plot of the implemented SNN's performance with other methods (WBCD).

The results presented in Table 2 have been obtained through software implementations of the author (Matlab, SNN) and from the specific literature, respectively [20]. Both datasets had been divided into training and test sets. The Matlab LM algorithm has been run for 50 epochs of 1500 cycles each.

Concluding, the implementation of a spiking neural networks presented in this article—with embedded soft-core microcontrollers—have proved to be more efficient in terms of FPGA hardware resource utilization, than other, fully parallel implementations, 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 [21–24].

This study reviewed the concepts of neuromorphic artificial neural networks and their possible digital hardware implementations. It proposed a framework on implementing these intelligent systems on reconfigurable circuits with embedded microcontrollers to demonstrate their superior performance in solving classification problems.

Table 2: Detailed comparison of the implemented hardware SNN's performance and structure with software implementation executed on single-core general purpose processor based PCs

					Precision (%)	
Algorithm	**Inputs**	**Hidden**	**Outputs**	**Iterations**	**Training set**	**Test set**
Fisher IRIS dataset						
Spike-Prop	50	10	3	1000	96.4	95.3
Matlab BP	50	10	3	2.6×10^6	97.9	95.8
Matlab LM	50	10	3	3750	98.8	95.9
$SNN_{Bakó}$	4	0	3	300	85	83.4
Wisconsin Breast Cancer dataset						
Spike-Prop	64	15	2	1500	97.6	97.3
Matlab BP	64	15	2	9.2×10^6	97.8	96.2
Matlab LM	64	15	2	3500	98.1	96.5
$SNN_{Bakó}$	9	6	2	250	90.2	89.5

The resulting systems show the potential of becoming useful tools in a wide range of possible applications in contemporary genetics and molecular biology.

Key Points

- The methodology presented can be a useful tool in many biology related applications.
- Hardware implementation oriented new learning rules and an input encoding mechanism are presented.
- The hardware implemented intelligent systems provide real-time and adaptive data processing.

FUNDING

The achievements presented in this article are part of the results of the research grant CNCSIS TD84/2008, directed by the author and funded by The National University Research Council of the Romanian Ministry of Education and Research.

References

1. Wulfram G, Werner W. *Spiking Neuron Models*. Cambridge: Cambridge University Press, 2002.
2. Roth U, Jahnke A, Klar H. Hardware requirements for spike-processing neural networks. In: Mira J, Sandoval F, (eds). *From Natural to Artificial Neural Computation (IWANN)*. Berlin: Springer, 1995;720–7.
3. Schaefer M, Schoenauer T, Wolff C, *et al*. Simulation of spiking neural networks – architectures and implementations. *Neurocomputing* 2002;**48**(1):647–79.
4. Bi GQ, Poo MM. Synaptic modifications by correlated activity: Hebb's postulated revisited. *Ann Rev Neurosci* 2001;**24**:139–66.
5. Bohte SM, Kok JN, La Poutré H. Spike-prop: error-back-propagation in multi-layer networks of spiking neurons. *Neurocomputing* 2002;**48**(1–4):17–37.
6. Deneve S. Bayesian inference in spiking neurons. In: Lawrence KS, Yair W, Léon B, (eds). *Advances in Neural Information Processing Systems*. Vol. 17. Cambridge, MA: MIT Press, 2005;353–60.
7. Rajesh PNR. Hierarchical bayesian inference in networks of spiking neurons. In: Lawrence KS, Yair W, Léon B, (eds). *Advances in Neural Information Processing Systems.* Vol. 17. Cambridge, MA: MIT Press, 2005:1113–20.
8. Richard SZ, Quentin JMH, Rama N, *et al*. Probabilistic computation in spiking populations. In: Lawrence KS, Yair W, Léon B, (eds). *Advances in Neural Information Processing Systems*. Vol. 17. Cambridge, MA: MIT Press, 2005:1609–16.
9. Delorme A, Gautrais J, VanRullen R, *et al*. SpikeNET: a simulator for modeling large networks of integrate and fire neurons. *Neurocomputing* 1999;**26–27**:989–96.
10. Natschläger T, Ruf B. Spatial and temporal pattern analysis via spiking neurons. *Network Comp Neural Syst* 1998;**9**(3): 319–32.
11. Bohte SM, Kok JN, La Poutre H. Unsupervised classification in a layered network of spiking neurons. *Proc IJCNN'2000* 2000;**IV**:279–85.
12. Omondi AR, Rajapakse JC, (eds). *FPGA Implementations of Neural Networks, Springer*. The Netherlands: Springer, 2006. ISBN-10 0-378-28485-0.
13. Snippe HP. Parameter extraction from population codes:a critical assessment. *Neural Comput* 1996;**8**(3):511–29.
14. Marchesi M, Orlandi G, Piazza , *et al*. Fast neural networks without multipliers. *IEEE Trans Neural Net* 1993;**4**(1):53–62.
15. Hikawa H. Frequency-based multiplayer neural network with on-chip learning and enhanced neuron characteristics. *IEEE Trans Neural Net* 1999;**10**:545–53.
16. Fisher RA. The use of multiple measurements in taxonomic problems. *Ann Eugen* 1936;**7**:179–88.
17. Mangasarian O, Wolberg W. Cancer diagnosis via linear programming. *SIAM News* 1990;**23**(5):1–18.
18. Sammon JW, Jr. A nonlinear mapping for data structure analysis. *IEEE Trans Comput* 1969;**C-18**:401–9.
19. Low KS, Krishnan V, Zhuang H, *et al*. On-chip genetic algorithm optimized pulse based rbf neural network for unsupervised clustering problem. *Lect Notes Comput Sci* 2006;**4222**:851–60.
20. Bohte S, Kok J, La Poutré H. Spike-prop: errorbackpropagation in multi-layer networks of spiking neurons. In: Verleysen M, (ed). *Proc Europ Symp Artificial Neural Networks (ESANN)*. D-Facto, 2000;419–25.
21. Maass W, Natschlager T, Markram H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput* 2002;**14**(11): 2531–60.
22. Schrauwen B, Van Campenhout J. Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic. In: Verleysen M, (ed). *ESANN'2006 Proceedings – European Symposium on Artificial Neural Networks, Bruges (Belgium)*. D-Facto, 26–28 April 2006.
23. Bako L, Brassai ST. Spiking neural networks built into FPGAs: Fully parallel implementations. *WSEAS Trans Circuits Syst* 2006;**5**(3):346–353.
24. Floreano D, Dürr P, Mattiussi C. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 2008;**1**: 47–62.