NEURAL NETWORK PARALLELIZATION ON FPGA PLATFORM FOR EEG SIGNAL CLASSIFICATION

László BAKÓ[#1], Ervin GYÖRGY-MÓZES[#], Sándor-Tihamér BRASSAI[#3], Lajos LOSONCZI[*4], László-Ferenc MÁRTON[#5]

[#] *NSRG - Neural Systems Research Group, Sapientia University Tirgu-Mures*

[1]lbako@ms.sapientia.ro

[3]tiha@ms.sapientia.ro

[5]martonlf@ms.sapientia.ro

[*] *Lambda Communication Ltd.*
Tirgu Mures, Str. Avram Iancu Nr. 37, Romania

[4]lajos@lambda.ro

ABSTRACT

This paper presents the design and FPGA implementation of an embedded artificial neural network system used for EEG signal classification. The hardware-software co-design techniques used in the designed process take advantage of the reconfigurable environment yielding a spiking neural network (SNN) that can differentiate between EEG signals presented as an image containing a pattern derived from the signals' Wavelet transform. Some modules of the system may use run-time partial reconfiguration for optimal FPGA resource utilization. The hardware platform used was an Opus Virtex 5 FPGA development kit with a built-in PowerPC 440 processor core. The developed SNN uses the processor core as well as the reconfigurable resources of the Virtex5 chip in order to implement a highly parallelized architecture optimized for the proposed task. Implementation details as well as performance analyses will be given in the paper.

Keywords: embedded system, FPGA, spiking neural network, parallelization, PowerPC, VHDL

1. Introduction

The use of modern technologies is becoming widespread in biomedical research. The development of embedded system techniques and their integration with novel signal processing methods has raised the possibility to devise structurally more refined, non-invasive EEG measurement systems and brain computer interfaces (BCIs) [1].

The development process considered the following priorities:

- improvement in signal acquisition, amplification and conditioning;
- tuning of adaptive systems by the use of embedded microcontrollers;
- implementation of smart active electrodes, that get "smarter" while reducing their size;
- real-time interpretation of the acquired signals via hardware implemented advanced methods.

The Smart Active Electrode (SAE) acquisition system developed by us is used, for the acquisition of the EEG signals that form the subject of the pattern recognition task presented in this paper.

2. EEG signals and brain-computer interfaces

Research areas of BCIs include the evaluation of invasive or noninvasive technologies to follow brain potential field activity. Electrical activity in the cerebral cortex contains important information to characterize brain function and is possible to study the role of individual cortical regions during event related processes, cognitive and motor tasks. Brain responses are, by definition cortical events translated into non-stationary EEG signals by active electrodes. We are involving special hardware and software procedures to handle the difficulties we met. Functional brain activity is associated with a variety of event-related changes in EEG spectra. The functional correlations of brain activities depend on the frequency bands in which they occur. Signal energy may increase in some frequency bands while it decreases in others, and the energy within one frequency band may increase at one time and decrease at another depending on its temporal relationship to the functional brain activation [2][3][4]. Usual EEG recording is performed with a

cap equipped with integrated scalp electrodes. EEG potentials, referenced to the average of the left and right ear lobes, are recorded at the standard fronto-centro-parietal standard locations F3, F4, C3, Cz, C4, P3, Pz, and P4 [5].

3. Hardware embedded spiking neural networks (SNN) used for signal classification

Spiking neurons (SNs) differ from conventional artificial neural network (ANN) models as information is transmitted by means of spikes rather than by firing rates. 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. When implemented on parallel hardware, ANNs can take full advantage of their inherent parallelism allowing specific units to be designed and added, that boost the computing speed. Implementing SNNs on reconfigurable hardware enables the identification of a number of challenges. Creating large-scale implementations of SNNs, particularly which operate in real time, and yet demonstrate biological plausibility (adaptability of the architecture) can become even more difficult. The presented implementation of a SNN, partly sacrifice the completely parallel nature of the design, by embedding hard-core processors, yet yielding a highly accelerated solution.

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 (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.

The values given as inputs to the network can be encoded into the synaptic values through delayed learning [6], [7], [8].

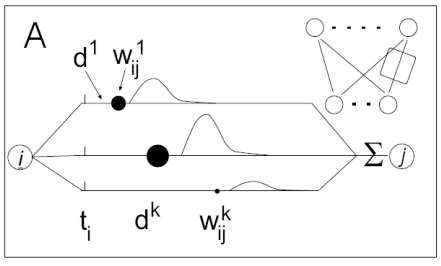

Fig. 1 – The theoretical concept of the implemented neural network

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 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:

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

according to [9], but this formula is unrealistic in terms of hardware implementations, hence a simpler method is needed. Each input neuron is allowed to emit a single spike during the encoding time frame.

4. Signal preparation for neural network input generation

During the first experiment, the subject of the EEG measurement was told to make timed movements. The signals were acquired from electrodes F7 and F3. The proposed goal was to determine the time of the movements by recognizing patterns in the signal with the implemented embedded neural network system.

The duration and the script of the experiment is pre-defined and it says that the subject should do the following for about one minute: 18/10 sec – staying still, 10 sec – making movement, 8/10 - staying still, 10 sec - making movement in opposite direction.

Our goal was to differentiate among the following three types of events by feeding the pre-processed signal into the embedded SNN: activity, noisy signal, no activity.

The first step of the pre-processing is the computation of Wavelet transform of the acquired EEG signal (Fig. 2), using a proprietary program. The same program marks those areas of the image representing the Wavelet transform of the signal (Fig. 3) that contain significant information. These areas need to be identified automatically by the developed SNN.

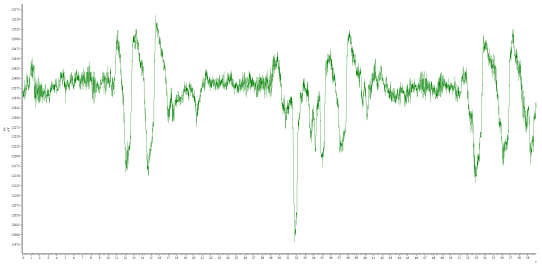

Fig. 2 – The acquired EEG signal

However, this image (Fig. 3) needs further processing in order to produce a set of images that can be fed into the SNN hardware.

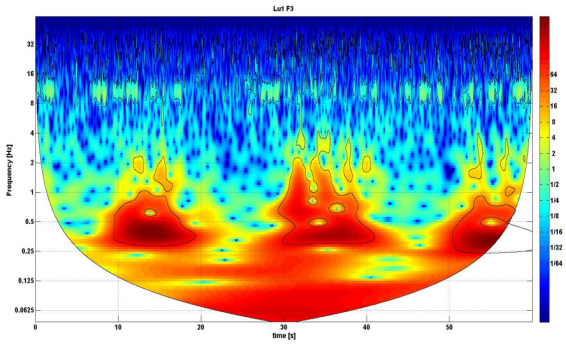

Fig. 3 – The wavelet transform of the measured EEG signal

This phase is also performed by a proprietary program of our research group and it yields the expected results – the images that will form the SNN's training and testing set – by performing the following processing steps:

1. The vector containing floating point values of the wavelet transform result is converted into an grayscale image (Fig. 4),
2. Background substraction for contrast enhancement,
3. Distribution of the measurement interval into equal time frames that will be converted into training images (64 x 64 pixels),
4. The images are pre-classified using the previously mentioned significance information in order to produce the expected output values of the SNN for all input images,
5. These images are saved into a screen-friendly format (Fig. 5, Fig. 6 and Fig. 7),
6. Image data is transmitted to the SNN.

The following figures present the pre-processed wavelet transform result and one example image of each of the three recognizable classes of the SNN training set.

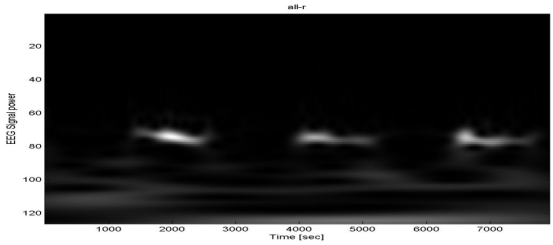

Fig. 4 – Grayscale image produced from the EEG signal's wavelet transform

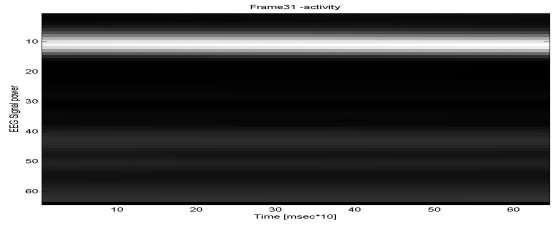

Fig. 5 – Image showing activity signal

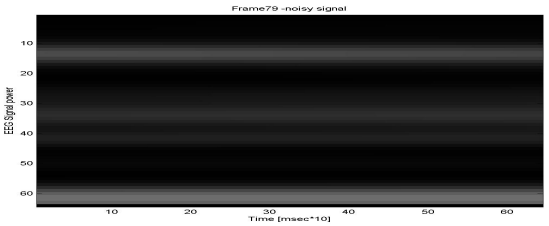

Fig. 6 – Image presenting a signal to be classified as noisy

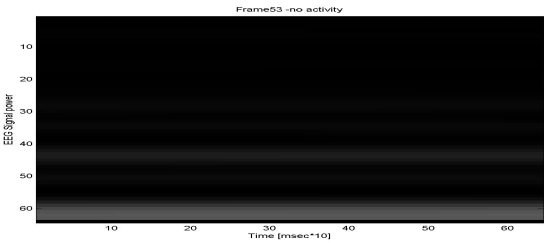

Fig. 7 - Image showing no activity signal

5. Hardware implementation of a SNN on a FPGA platform

In order to make best use of the Opus Development board featuring a Xilinx Virtex™-5 FX30T FPGA with embedded PowerPC processor the project has been divided into components implemented as software run on the PowerPC and highly parallelized hardware components that use the reconfigurable fabric of the FPGA.

Structurally, the implemented SNN can be divided into three major components: the input neurons, the synapses and the neuron bodies or somas. Synapses and somas form the output layer neurons.

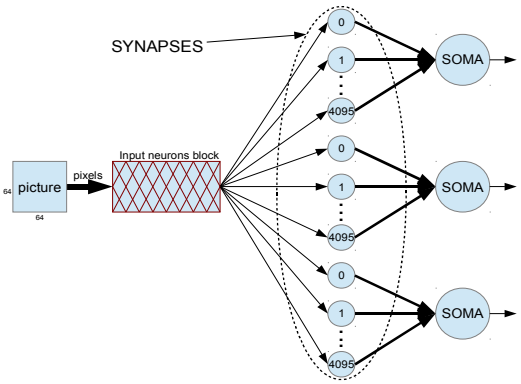

Fig. 8 –The structure of the FPGA implemented SNN

As the block diagram in Fig. 8 above shows, the role of the input neurons is to encode the input images into time-delayed spike-trains that will propagate through the network and form a spike at one of the outputs. The functional component design using the Xilinx EDK environment yielded the structure presented in Fig. 9, where the MyIP module is the one implementing the SNN specific circuitary.

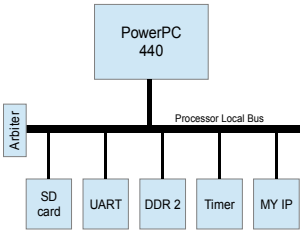

Fig. 9 – The main modules of the Xilinx EDK project that implements the SNN

6. Computing image data into spike trains as input to the SNN embedded in the FPGA

For each of the 64x64 pixels of the training set images a numerical value has to be generated, representing a grayscale level on eight bits. These have been computed based on the wavelet transform of the measured EEG signal, using a proprietary C program, and then saved into files either on the DDR RAM memory modules or on the SD card attached to the Opus board.

7. Implementation of the input neurons and pulse stream generation

After studying [10], [11], [12] 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=37 is assigned to neurons with weaker stimulation (Fig. 10).

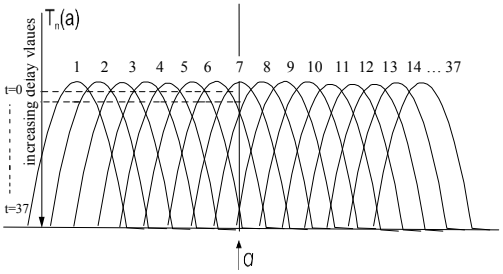

Fig. 10 - The method of encoding the input variables into delayed spikes

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 expensive way to do this, in terms of number of neurons needed, is to encode each input with 1-D, independent receptive fields [14]. Since the interest is in two-dimensional 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.

A triangular approximation of the Gaussian activation functions of the input neurons is much more suitable for the aimed hardware implementation, due to reconfigurable resource costs. On the other hand, though, these triangular functions need to be designed with maximal overlapping areas in order to yield the highest possible number of activated input neurons for each input value. A scheme with 37 input neurons (triangles) has been devised for the proposed application. All these input neurons and their activation functions are computed by the C program that runs on the PowerPC.

8. The parallelized output layer of the embedded SNN

The output layer of the SNN is formed by the three somas with their 64x64 synapses, yielding a total of 12288 synapses units, all implemented as parallel cores in the reconfigurable fabric of the FPGA. All these units were designed in VHDL with the use of generic parameters in order to guarantee scalability. As Fig. 13 shows, the hardware implemented section of the SNN has three main functional units.

The Control Unit (CTRL) is the system manager, coordinating all stages of the computation. The NETWORK unit contains all the somas and synapses of the SNN with their weight values and those temporal moments when they contribute to the appearance of an output spike.

The LEARN module is responsible for the neural learning process employing a finite-state machine. The dynamic run-time reconfiguration capability of the Virtex FPGA can be used to inter-change the CTRL and LEARN units – therefore saving precious resources – since they do not run their algorithms in parallel.

The algorithm used to build the finite-state machine of the CTRL unit is presented in Fig. 12, which goes through the following states:

- *Ready:* Waits while data is being transmitted through the PLB bus. When EN is active the machine switches to the *Data_ready* state otherwise stays in the current state.

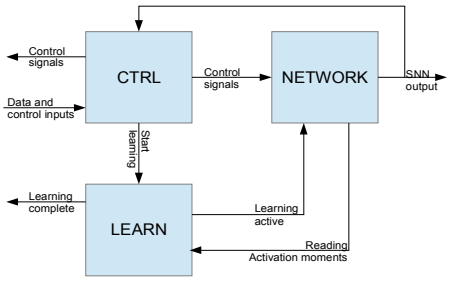

Fig. 11 – Modular structure of the implemented SNN system

- *Data_ready:* if the memory address counter reaches 4096 – meaning that all spike trains have been generated – , then switches to the *TH_Test* state or else to the *Bit_Test* state,
- *Bit_Test:* Watches for logical 1 values in the impulse set. Upon finding such a value on a particular position it computes – uses an offset – the memory address of the corresponding weight to be read.
- *Read:* The BlockRAM memories that store the weight values and the pulse trains are enabled,
- *OR:* The OR and SOMA units are clocked and the yielded spike train will give the enable signal to the synapse that needs to activate at this moment, then switches to the *Write* state,
- *Write:* The generated spikes are written to a BRAM memory, then goes back to the *Bit_Test* state,
- *TH_Test:* This state is the one where the SOMA modules are activated, as they receive and accumulate the weight values from the active synapses. Also tested are the threshold and membrane potential value to decide whether the cell will emit a spike or not and the time-step counter that runs until the value of 37, according to the lengths of the input spike train.
- *Out_Test:* The number of the activated SOMA is saved to a variable for the LEARN module that will use this as a reference in the learning rules.

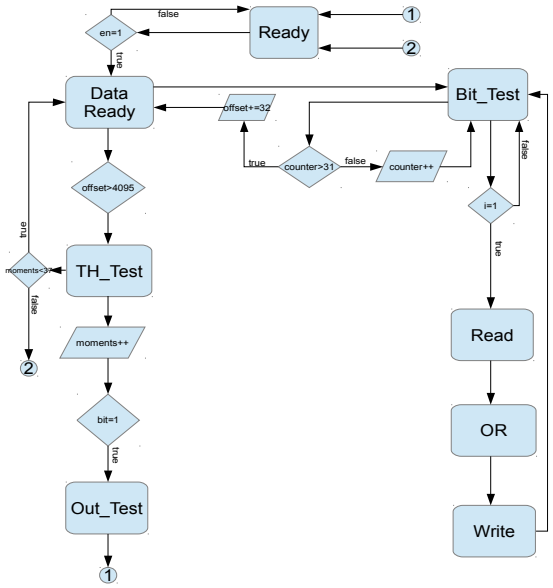

Fig. 12 –State diagram of the CTRL unit

9. Hardware structure of the embedded SNN

The actual components of the hardware built SNN are subunits of the NETWORK unit. The structure of this unit is presented in the Fig. 13 below. There are three ***MEM_WEIGHT*** memory modules, storing the synapse weight values for each of the three somas. These memory units have been generated using the Xilinx EDK's Core Generator tool and will yield a weight value to the internal data bus

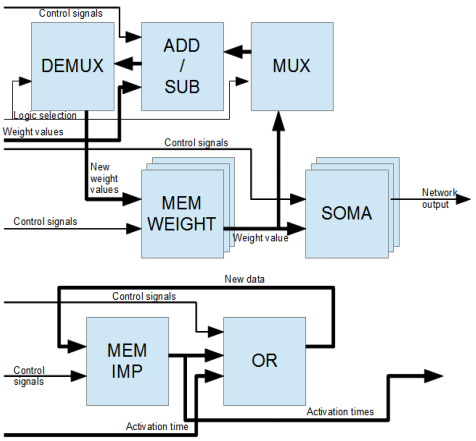

Fig. 13 –Functional modules of the SNN's hardware components

(10 bits) when addressed (on 12 bits) and enabled or will save one from the same bus when the learning mechanism modifies the strength of a synapse. Initial weight values were pre-computed using a C program and pre-loaded into the *MEM_WEIGHT*.

The three cell body or ***SOMA*** modules are responsible for the spiking neuron's output generation. The most important parameters defined here are the membrane potential (MP), threshold potential and the resting potential values. When the enable signal is received active and there is a clock transition, the SOMA modules will update the membrane potential parameter values, by summing the incoming weight values of the currently activated synapses. The threshold potential (THP) and the resting potential (RP) values are defined as follows:
THP=weight average nr. of weights * spike average * α%*
*RP=weight average * nr. of weights * spike average * β%,*
where the factors α=90 and the β=10.

If the sum of the updated MP and the RP exceeds the value of THP, then the SOMA will activate (fire), emitting an output spike (output='1'). The ***OR*** module will merge these new generated output spikes with the previously computed data and feed these together into the ***MEM_IMP*** module that stores the soma output values and the time-step number when they were generated. Since the time-frame in this application has been set to 37 (equal to the number of input value encoding input neurons) the data bus width of the MEM_IMP module is also 37 bits. The ***ADD/SUB*** module is used during the learning process, to change the weight values that either needs to be increased or decreased. It will add or subtract a preset value to/from the received weight value (on the 10 bit wide weight bus) depending on the state of a control signal and the presence of the enable signal and the clock transition. Since the learning rules are applied through the ADD/SUB module to the weights of a single soma there was need for a ***MUX*** module to select which MEM_WEIGHT module has to be addressed at a given step of the learning process.

10. The learning process used by the SNN

The learning rules devised for this application is another original contribution of the authors of this paper. The algorithm can be considered as a modified, supervised version of the Hebb learning rule. In this application it has been tuned to fit the parameters of the built SNN. The 37 input neurons yield the 37 step time-frame. The devised learning rule is active in only eleven of these steps. In the time-step *n*, when MP≥THP, the weights are increased by a ΔW value (currently ΔW=60). This time-step *n* is always set to be the sixth step of the eleven when the learning is active. In the rest of the eleven steps - the five steps preceding and following *n* - will add or subtract, respectively, a value to/from the weights, value that decreases with $1/6^{th}$ of ΔW by each time-step.

This process is presented in detail in table 1.

As mentioned before, the learning process is implemented in the LEARN unit, consisting of an elaborate finite-state machine with the state diagram presented in Fig. 14.

The operation of this unit is best to be presented via the job of each of the states that are the following:

- *Idle* – Waits for the signal that activates the learning process.

Table 1 – Rules of the modified Hebb learning

Synapses	*Axon*	*Target*	*Time step*	*Operation*	*Weight changes*
0	0	0	--	--	--
0	0	1	--	--	--
0	1	0	--	--	--
0	1	1	--	--	--
1	0	0	--	--	--
1	0	1	--	--	--
1	1	0	5<	+	ΔW-5*(ΔW/6)
1	1	0	4<	+	ΔW-4*(ΔW/6)
1	1	0	3<	+	ΔW-3*(ΔW/6)
1	1	0	2<	+	ΔW-2*(ΔW/6)
1	1	0	1<	+	ΔW-(ΔW/6)
1	1	0	=	+	ΔW
1	1	0	1>	-	ΔW-5*(ΔW/6)
1	1	0	2>	-	ΔW-4*(ΔW/6)
1	1	0	3>	-	ΔW-3*(ΔW/6)
1	1	0	4>	-	ΔW-2*(ΔW/6)
1	1	0	5>	-	ΔW-(ΔW/6)
1	1	1	--	--	--

- *Check_Out* – Verifies if the outputs of the network match the prescribed values, where they do not, the memory access signal is set, then it switches to the *Position* state. Where they do match, the machine goes back to the *Idle* state, therefore ending the current learning cycle.
- *Position* – The time-step counter (that increments up to the end of the time-frame) will determine which of the Hebb rules will be applied (detailed in table 1) then it sets the corresponding ΔW value.
- *Read_Imp* – The clock, address and enable signals of the *MEM_IMP* module are generated in this state in order to read the spike streams for this module. While the address counter is less than 4096 it will step back to the *Position* state.
- *Read_Weight* – The *MEM_WEIGHT* module is activated in order to read the weight from the synapse selected by the address counter. It is followed by the *Add/Sub_Weight* state.
- *Comp* – The current time-step value is compared to the value read from the spike train, if

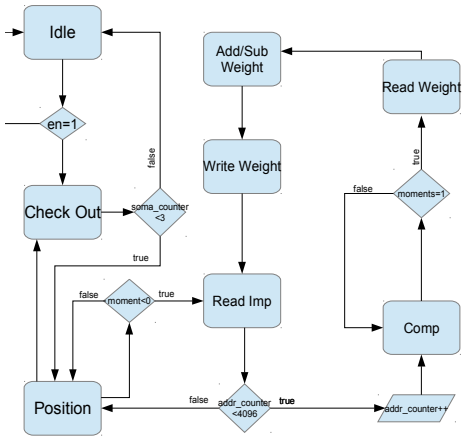

Fig. 14 – State diagram of the LEARN unit

both are active ("1"), than the corresponding weight needs to be read (go to *Read_Weight* state) else the next impulse is read in the *Read_Imp* state.

- *Add/Sub_Weight* – This is the state where the selected weight is actually modified, according to the rules table, by either increasing or decreasing its value, therefore strengthening or weakening the efficiency of the respective synapse.
- Before returning to the *Read_Imp* state the modified weight needs to be saved by activating the *MEM_WEIGHT* module.

All components of the implemented hardware SNN have been verified and validated to function properly, via simulations.

11. Results and conclusions

The presented project has some major achievements in embedding a fairly large SNN into a Xilinx Virtex FPGA circuit by finding a good balance between the software implemented part (in the PowerPC processor core of the Virtex) and the massively parallelized components occupying the reconfigurable fabric of the circuit. Summarizing the results we can affirm that the hardware/software co-design technique used has yielded the following outcome. The input neurons of the SNN application implemented on FPGA for EEG signal classification were implemented on the 400 MHz PowerPC core embedded into this circuit. This software has multiple tasks, but the most important is the conversion of the input image data into encoded spike trains, a process that takes 2.5 milliseconds. Besides this task the PowerPC program (written in C) is also responsible for reading the input image data from the SD card attached to the FPGA circuit, a process that takes up 47 milliseconds per image, while encoding one image

to spikes takes 23.39 milliseconds. These values might seem large but these processes only take place once, immediately after the FPGA configuration, therefore do not have an impact on the overall performance of the SNN system. Finally, the initial weight values are computed in the software part of the project. The generated spike trains are then transferred to the hardware implemented (using VHDL) part of the project via the PLB bus connected to the PowerPC processor. The output neurons with their synapses and somas are all implemented as units working in parallel. The control unit coordinates the various memory modules and finite-state machines that contribute to the successful implementation of the supervised version of the Hebb learning rule.

As the hardware resource utilization of the project presented in table 2 shows, there is plenty of space left in the reconfigurable fabric of the FPGA to extend the project for true color image classification.

Table 2 – Resource utilization of the SNN project

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Slice Registers	334	20480	1%
Slice LUTs	562	20480	2%
Fully used LUT-FF pairs	288	608	47%
Bonded IOBs	77	360	21%
Block RAMs / FIFOs	9	68	13%
BUFG/BUFGCTRLs	4	32	12%

The achieved performance has yielded an output computation time for a grayscale 64x64 pixel image of 2.6ms while a full learning cycle took 0.75ms.

Table 3 – Classification results of the SNN

THP (% of max MP)	**Cycle nr. / image** (60 images)	**Number of images classified correctly**
50	50	30
50	100	29
50	200	32
70	50	28
70	100	31
70	200	32
90	50	22
90	100	27
90	200	28

The classification accuracy assessment of the implemented embedded SNN system has been made using a few set of images derived from EEG signals measured by our research group. These sets of about 100 images have been divided into training and a testing group. The SNN has been trained with input data generated from the first group of images then tested for data from the second group. Table 4 gives a few examples of the yielded results. It can be concluded that the classification accuracy is yet to be enhanced by fine-tuning the learning rules and also by using other training image sets with significantly higher number of images.

Acknowledgement

This work is part of the project funded by the Romanian National Authority for Scientific Research, grant No. 347/23.08.2011.

References

[1] Karl H, Willig A, Wolisz A. *Wireless Sensor Networks*, Eds. Springer Verlag Berlin, 2004, 1-17.

[2] Krausz, G., Scherer, R., Korisek, G. and Pfurtscheller, G. *Critical Decision-Speed and Information Transfer in the 'Graz Brain–Computer Interface'*; Applied Psychophysiology and Biofeedback, Vol. 28, No. 3, September 2003.

[3] Miller, L.C., Ruiz-Torres, R., Stienen, A.H.A. and Dewald, J.P.A. *A Wrist and Finger Force Sensor Module for Use During Movements of the Upper Limb in Chronic Hemiparetic Stroke*, IEEE Transactions on Biomedical Engineering, No.9, 2009.

[4] Johansson, R.S. and Flanagan, J.R., *Coding and use of tactile signals from the fingertips in object manipulation tasks*, Nature Reviews Neuroscience, Volume 10, May 2009, 345.

[5] Piipponen, K.V.T., Sepponen, R. and Eskelinen, P., *A Biosignal Instrumentation System Using Capacitive Coupling for Power and Signal Isolation*, IEEE transaction on biomedical, Vol. 54, No. 10, October 2007.

[6] Natschläger, T., Ruf, B. *Spatial and temporal pattern analysis via spiking neurons*, Network: Comp. Neural Systems 9 (3), 1998, pp. 319–332.

[7] 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.

[8] Edited by Omondi, A.R.& Rajapakse, J.C., *FPGA Implementations of Neural Networks*, Springer, 2006, ISBN-10 0-378-28485-0(HB).

[9] Rajesh P. N. Rao. *Hierarchical bayesian inference in networks of spiking neurons.* In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, Advances in Neural Information Processing Systems 17, MIT Press, Cambridge, MA, 2005, pp. 1113–1120.

[10] Maass, W., Natschlager, T. and Markram, H. *Real-time computing without stable states: A new framework for neural computation based on perturbations*. Neural Computation, 14 (11), 2002, pp. 2531–2560.

[11] B. Schrauwen, J. Van Campenhout, *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, ISBN 2-930307-06-4.

[12] D. Floreano, P. Dürr, C. Mattiussi, *Neuroevolution: from architectures to learning*, Springer-Verlag, Evolutionary Intelligence, 2008, 1, pp. 47–62, DOI 10.1007/s12065-007-0002-4.