Proceedings of the 8th WSEAS Int. Conference on Automatic Control, Modeling and Simulation, Prague, Czech Republic, March 12-14, 2006 (pp261-266)

Hardware spiking neural networks: parallel implementations using FPGAs

LÁSZLÓ BAKÓ, SÁNDOR-TIHAMÉR BRASSAI
Electrical Engineering
Sapientia – Hungarian University of Transylvania
540053 Tîrgu Mureş, P-ţa Trandafirilor 61.
ROMANIA
lbako@ms.sapientia.ro, tiha@ms.sapientia.ro

Abstract: - Neuromorphic neural networks are of interest both from a biological point of view and in terms of robust signaling in noisy environments. The basic question however, is what type of architecture can be used to efficiently build such neural networks in hardware devices, in order to use them in real time process control problems. In this paper a novel, hardware implementation friendly, "pulse reactive" model of spiking neurons is described. This is used then to implement a fully connected network, yielding a high degree of parallelism. The modular neuron structure, acquired signals and a process control application are given.

Key-Words: Neuromorphic neural networks, Spiking models, Simulation, Parallel FPGA implementation.

1 Introduction

Artificial neurons have been modeled in the last years from a different approach, by trying to mimic the most important features of real neurons and to achieve further aspects of computation like parallelism and learning. This way, building biologically inspired neuron models, then using these to construct so-called neuromorphic artificial neural networks, has become an important path to follow up, due to their big potential in developing timing features of neural computation [2].

An artificial neural network (ANN), though, is a massively parallel, distributed processor made up of simple processing units. It resembles the human brain in two respects: knowledge is acquired through a learning process, synaptic weights are used to store the knowledge [3]. Among other features, ANNs provide nonlinearity, are universal approximators, adaptable (by weights and/or topology), and intend to be neurobiologically plausible.

On the other hand, the impressive number of neurons in the mammal cortex and the vast connectivity between them (up to 10^4 connections per neuron) are very difficult to mimic either in software or hardware. Software simulation of a network of thousands of neurons might take hours to complete on a fast Pentium based computer. In a previous work [1] a novel simulation environment has been presented, which dealt with the spiking behavior of neurons, using a special neural model. Using this environment any 3D spiking neural network can be built either giving the architecture neuron-by-neuron or layer-by-layer. Feedbacks are also supported allowing recurrent networks to be built.

In Section 2 of this paper a special spiking neural model will be presented, suitable for FPGA implementation. The details of the implementation are given in Section 3 followed by the future challenges and prospects. The final section concludes by presenting measurements and experimental results.

2 Neural Model

Many neuron models, such as perceptron or radial basis functions, use real values as inputs and outputs, processed using gaussian or other continuous functions. In contrast, biological neurons process spikes: as a neuron receives input spikes by its dendrites, its membrane potential increases following a post-synaptic response. When the membrane potential reaches a certain threshold value, the neuron fires, and generates an output pulse through the axon. One of the best-known biological models is the Hodgkin and Huxley model (HHM) [4], which is based on ion current activities through the neuron membrane.

The most biologically plausible models are often not the best suited for computational implementations. This is the reason why other simplified approaches are needed. The leaky integrate and fire model (LIFM) [5] is based on a current integrator, modeled as a resistance and a capacitor in parallel. Differential equations describe the voltage given by the capacitor charge, and when a certain voltage is reached the neuron fires. The spike response model

(SRM) [5] offers a response that resembles that of the LIFM model, with the difference that the membrane potential is expressed in terms of kernel functions [5] instead of differential equations.
Spiking neuron models process discrete values representing the presence or absence of spikes. This fact allows a simple connectionism structure at the network level and a striking simplicity at the neuron level. However, implementing models like SRM and LIFM on digital hardware is largely inefficient, wasting many hardware resources and exhibiting a large latency due to the implementation of kernels and numeric integrations. This is why a functional hardware-oriented model is necessary to achieve fast architectures at a reasonable chip area cost.

2.1 The proposed neuron model

An artificial neural network (ANN) is a parallel and distributed network of simple nonlinear processing units interconnected in a layered arrangement. Parallelism, modularity and dynamic adaptation are three computational characteristics typically associated with ANNs. FPGA-based reconfigurable computing architectures are well suited to implement ANNs as one can exploit concurrency and rapidly reconfigure to adapt the weights and topologies of an ANN.
FPGA realization of ANNs with a large number of neurons is still a challenging task because ANN algorithms are usually "multiplication-rich" and it is relatively expensive to implement multipliers on fine-grained FPGAs. By utilizing FPGA reconfigurability, there are strategies to implement ANNs on FPGAs cheaply and efficiently.
One of these strategies, is to build them without using any multiplicator circuits. Most multiplier-like tasks are fulfilled by sequential counter logic circuits. This makes our approach more cost-efficient than the hardware built, perceptron based ANNs. Due to the high level of parallelism achieved in this manner speed enhancement is also obvious.
Our simplified model presented in a previous paper [1], uses the following concepts: 1 - membrane potential; 2 - resting potential; 3 - threshold potential; 4 - postsynaptic spike; and 5 - presynaptic spike (see Fig. 1). A spike is represented by a pulse. The model is implemented as a Moore finite state machine. Two states, operational and refractory, are allowed. The spiking pulses can be of positive potential (Stimulating Spiking Pulse – SSP) or of negative one (Inhibitory Spiking Pulse – ISP) [10]. Assume that two presynaptic neurons (j=1,2) send SSP towards a postsynaptic neuron: j=1 neuron is firing in the moments $t_1(1)$, $t_1(2)$,…, $t_1(n)$ and j=2 in the moments $t_2(1)$, $t_2(2)$,… $t_2(n)$ (generally $t_j(f)$).

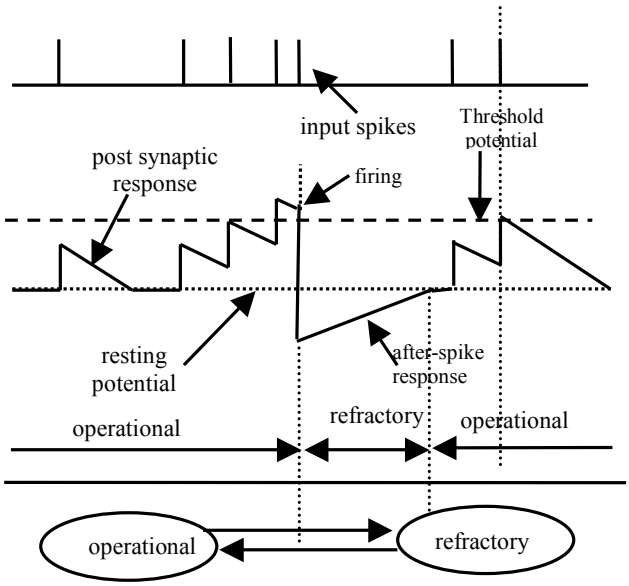

Fig. 1. Dynamics of the implemented neural model

Hence, during the operational state, the membrane potential is increased (or decreased) by $\varepsilon_{i1},\varepsilon_{i2},\ldots\varepsilon_{ij}$ each time a pulse is received by an excitatory (or inhibitory) synapse, and then it decreases (or increases) with a constant slope until the arrival to the resting value (Fig. 1.). If a pulse arrives when a previous postsynaptic potential is still active, its action is added to the previous one. This pulse reactive model (PRM) is, maybe, the most appropriate method to model spiking neural networks (SNN), the task being to evaluate the firing moments of firing neuron (FN) i ($t_i(1)$, $t_i(2)$,…$t_i(n)$) as a function of the firing moments of j presynaptic FNs ($t_j(1)$, $t_j(2)$,… $t_j(n)$).
The membrane potential of a cell body (soma) of the neuron number i is:

$$u_i(t) = \sum_{i,j} w_{ij} . \varepsilon_{ij}(t - t_j^{(f)}) \qquad (1)$$

which expresses the contribution of j presynaptic FNs, generated before the moment t.
The PRM handles also with a damping factor $\eta(t-\hat{t}_i)$, which is a function of $\hat{t}_i$-the moment of the generation of the last spiking pulse of FN i, before the time t. If $t-\hat{t}_i < 0$ or $t-\hat{t}_i > 0$ and high enough, the damping factor is zero. If $t-\hat{t}_i > 0$, but of small value (that means the i FN just has generated a spiking pulse), the damping factor has strong negative value and inhibits the FN i to generate a new spiking pulse. Comparing to the Hodgkin-Huxley model [2, 5], the pulse reactive model (PRM) is more suitable to computer analysis, as there are no differential equations handled in the model. Choosing appropriate values for $\eta, \vartheta, \varepsilon$ functions, the PRM could model quite well the dynamics of the natural, biological neurons. Our

model uses a coding algorithm, where each spiking pulse has a separate importance, similar to the bits in the computers. As the time is continuous, a single spiking pulse could include more information than a single bit, as the arriving time t could encode the analog value (t-T), when T is a reference time value. In this respect a FN acts as a coincidence detector, i.e. will generate a spiking pulse only if a number great enough of SSPs will arrive to the neuron cell (soma), almost simultaneously. If all the transmission delays Δ_{ij} between the FN *i* and the presynaptic *j* FNs are identical, then the FN *i*, which has a high threshold value ϑ_i, will fire only if all presynaptic *j* FNs will have the firing moments $t_j(f)$ very close to each other (practically are simultaneous). If the Δ_{ij} are different for several *j* presynaptic neurons, the FN *i* will fire only if the presynaptic neurons will comply with a certain scheduling in firing times: *i* will fire if for any *j*, $t_j \approx T-\Delta_{ij}$ [6].

3 Spiking Neural Network Built Into a FPGA-based System

Based on the mathematical model and the results of the simulated network several hardware implementations of spiking neural networks were developed.

Two main parts of the neuron model, the synapse and the soma, were designed and developed separately. The neural model's specific properties allow the inputs and outputs to be easily given as binary values (pulses). This was one of the reasons, which led to the digital hardware implementation. The device used was a XESS XSA3S1000 prototyping board with a XILINX XC3S1000 Spartan III FPGA (1000k logic gates). Beside the FPGA chip there is a CPLD IC for the PC parallel port connection interface, a 32Mb SDRAM, a 2Mb Flash RAM memory module present on the prototyping board, the FPGA being driven by a programmable oscillator with the maximum frequency of 100 MHz.

3.1 Strategies in developing optimal FPGA implementations of adaptive artificial neural networks

The model used is based on the spiking behavior of natural neurons, but does not deal with the ion channel dynamics of biological cells, only with the signal flow between the units of a neural network, explicitly the row of action potentials in our case.

In contrast with activity rate encoded networks, the SNN's developed by us are encoding the precise timings of the spike arrivals, not the density rate of these. Thus, information is being transmitted among neurons solely by action potentials. The neural model presented in [1] and briefed in the previous section, has been developed into a digital model, which defines *q* feeding dendrites. The input on these dendrites reach the cell body (soma) through synapses, multiplied by some values, modifying the membrane potential of the soma. Eventually this will result in the activation of the neuron, generating an axonal action potential (output spike). According to the solution suggested in [1] the feeding dendrites can be side-connected in order to enhance the synchronization of neurons from a certain layer. Other dendrites could be inhibitory inputs, thus lowering the membrane potential of the soma. The inhibitory effect can be implemented alternatively by building a single inhibitory module, with effect over the entire network. This module was developed to receive the axonal spikes of all or majority of the neurons in the network. After processing this module sends an equally distributed inhibitory signal to the connecting neurons. All connections to and from the inhibitory module have the same weight, hence it can be built at a low implementation cost with hardware elements, given, that only a few parameters need to be handled.

3.2 Experimental inhibitory module in the hardware built neural network

The output of any activated neuron of the network will increment a counter in the IHM. At the end of a time-step (the operation of the hardware SNN is divided into time-steps [1]) the value of the counter is multiplied with an inhibitory weight. This value is stored in a register, then the counter is reset. Hence, at the end of the time-step, this value will reflect the intensity of the network's neural activation during the time-step. In the next step one can use this value as an inhibitory input of the network. In this respect the calculated value will be added to a global inhibitory dendrite potential , DP^{I}_{glob} which will decrease with a slope of γ_I in each time-step. Considering a simple case, when the distribution of the inhibition is even on the entire network, then all the neurons contributing to the inhibition are part of a N_{Inhib} set. The active neurons of this set contribute to the inhibition process at a rate given by a ω^{I}_{Glob} weight. Under these circumstances an inhibitory dendrite DP^{I} potential does not need to be computed for each neuron, only the global value given by the following expression:

$$DP^{I}_{glob}[n] = \omega^{I}_{Glob} \cdot \sum_{j=1}^{j \in NI_{Inhib}} y^{I}_{j}[n-1] + \gamma_{I} \cdot DP^{I}_{glob}[n-1] \quad (2)$$

where y^{I}_{j} is the static threshold potential of the neurons. The inhibitory module is not fully developed and tested yet, so it was not implemented in the hardware built SNN presented in this paper.

3.3 FPGA implementation of the synapse with modified learning algorithm

Considering the behavior of a cerebral synapse, the main function of the circuit developed by us as the artificial synapse is a pulse multiplier, although it doesn't contain any real multiplicators. As the architectural schematic in Fig. 2 presents, the synapse is also made up by two – structurally and functionally – separate units. The structure of this module has been redesigned since [1], yielding a much more compact form. During this optimization, all subunits were constructed using only native (primitive) basic elements of the Xilinx FPGA (one synapse consumes only 17 slices of a XC3S1000 after Place and Route (PAR) phase).

The implemented learning algorithms were also

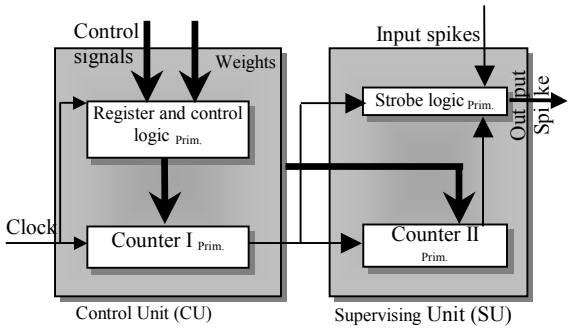

Fig.2. The synapse

changed and diversified. There are four types of weight-adaptation rules, which were selected according to their neuro-phisiological plausibility and fitness for hardware implementation. In other words, these rules secure the neuromorphic nature of these artificial models, at functional level, too, by following suit with the local adaptation mechanisms most often found in the neurons of mammals brain [6]. Some restrictions limit the capability of these rules in tuning the weight values:

- the weights may only increase up to a maximum level,
- the weights may not change sign,
- the weights were initialized with fixed or random values,
- the weight may be altered in each time-step, according to the following relation:

$$\omega_{ij}^{t} = \omega_{ij}^{t-1} + \eta \Delta \omega_{ij} \quad (3)$$

where η is the learning rate and $\Delta\omega_{ij}$ is defined by one of the learning rules presented next.

The simple Hebb rule: the efficacy of the synapse can only increase is a certain correlation is detected between the activity of the x_j presynaptic and the y_i postsynaptic neurons (both are active during a time frame) [6]. The following expression determines how the weight change in this case:

$$\Delta \omega_{ij} = (1 - \omega_{ij}) x_j y_i \quad (4)$$

Postsynaptic rule: in addition to the simple Hebb rule, the synaptic strength will decrease if postsynaptic element is active but the presynaptic is:

$$\Delta \omega_{ij} = (-1 + x_j) y_i + (1 - \omega_{ij}) x_j y_i \quad (5)$$

Presynaptic rule: complementary to the postsynaptic rule, it states, that a synapse is weakened also, when the presynaptic neuron is active but the postsynaptic one is not:

$$\Delta \omega_{ij} = \omega_{ij} x_j (-1 + y_i) + (1 - \omega_{ij}) x_j y_i \quad (6)$$

Covariance rule: the synapse efficiency grows if the difference between the activation of the two neurons is less then half of the maximum activation value, otherwise diminishes. This means, that the synapse between two neurons will get stronger, if the neurons have similar activity levels. [10]:

$$\Delta \omega_{ij} = \begin{cases} (1 - \omega_{ij}) F(x_j, y_i) & \text{if } F(x_j, y_i) > 0 \\ \omega_{ij} F(x_j, y_i) & \text{otherwise} \end{cases} \quad (7)$$

where, $F(x_j, y_i) = \tanh\left(4\left(1 - |x_j - y_i|\right) - 2\right)$ is the scale of the difference between the pre- and postsynaptic activities. If the difference is greater or equal then 0.5 (half of the maximal activity), then $F(x_j, y_i) > 0$ and $F(x_j, y_i) < 0$ if the difference is less then 0.5.

3.4 Optimization of the soma module

The soma module can be considered as the Central Processing Unit of the neuron model.

The architecture of the soma is built on four modules (Fig. 3). The main task of the soma is to calculate the membrane potential using the input spikes from the synapses, compare it to the threshold potential (THP) and if the latter is the smaller one it has to emit an axonal spike. On the falling edge of the clock signal, the Synaptic Inputs (SYNIN) unit of the soma reads the output values of the synapses and sums these into an input value. The Membrane Potential Computing Unit (MPCU) unit adds this input to the previous value of the membrane potential. If there were no input spikes, then the MP decreases with a constant slope, down to the resting potential (4~5% of the maximal MP). The MPCU module also contains a time-frame creation counter

used to limit the time while a certain number of input spikes have to arrive in order to cause an

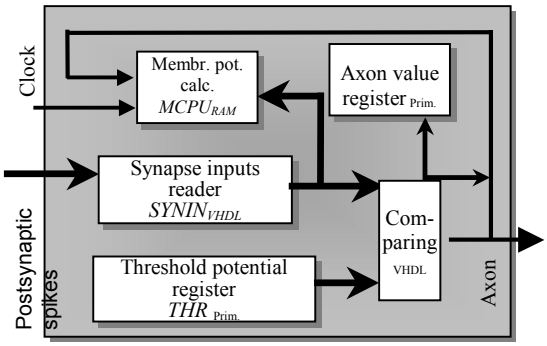

Fig. 3. The hardware-implemented soma

axonal spike. (Only excitatory synapses were implemented). The updated MP is then compared to the adjustable threshold-potential. If the required criteria is met (MP>THP), an axonal spike is emitted and the neuron is placed into hyperpolarizational phase, when the MP is set below the resting potential (zero). The latest soma module is capable of receiving input spikes through up to 32 synapses. The representations of the MP and THP were carefully chosen to be on eight bits. Without using RAMs to store parameters this assures the prospect of building larger SNN with the limited resources of the FPGA system. The weight values are stored on four bits. The structural optimization of the soma, lead to a mixed setup. The MCPU has been incorporated into the BlockRAM modules, the SYNIN and the comparison modules stayed as VHDL code, while the storage units are instantiated primitives. Consequently, the number of slices used by the soma after PAR dropped to 15 from ~400 [1].

3.5 Challenges and prospects of the hardware implementation

The compromise between giving up the fully parallel hardware implementation and the resource appetite of this approach has been raised at each step of the design process. The question, that one can ask is whether it is worth implementing these SNNs in fully parallel fashion, or serializing the system – even only certain parts of it – is the better choice, in order to be able to build larger networks.

Most of the recent hardware implementations of SNNs [7, 8, 9] build only one neuron using ASIC or reconfigurable circuits or use auxiliary devices. All in all, some compromise is unavoidable, since it is impossible, now, to build a similar digital model with such detail and precision using FPGA than with an ASIC containing a single spiking neuron. Despite the low cost implementation and the relatively high FPGA resource utilization, the presented system has shown its reliability and high performance even in its former setup [1].

4 Description of the Implemented SNN. Experimental Setup and Results

The experimental setup consists of two parts: a software simulation for a spiking neural network using the environment developed and presented in [1], and its respective validation on a FPGA.

Following a successful benchmark implementation presented in [1], a character recognition system, the newly proposed application, based on the improved neural model, is a simple process control system, implementing a gas-yield neural-controller for a nonlinear system (very low yield input of Ar, N, C_2H_2 gases into a vacuum chamber).

4.1 Description of the network and the experimental environment

The novel, FPGA implemented SNN consists of 32 cell bodies with thirty-two synapses each. The neural network has 32 inputs, 24 neuron in a hidden layer and 4 neurons in the output layer (Fig. 4.) with several control signal inputs. The constructed experimental system, consisting of: the XESS XSA3S1000 FPGA development board, an OMEGA CIO-DAS08 data acquisition board (3x8 bit digital I/O ports) on a

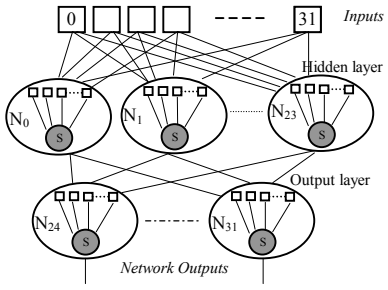

Fig. 4. – Topology of the FPGA implemented spiking neural network

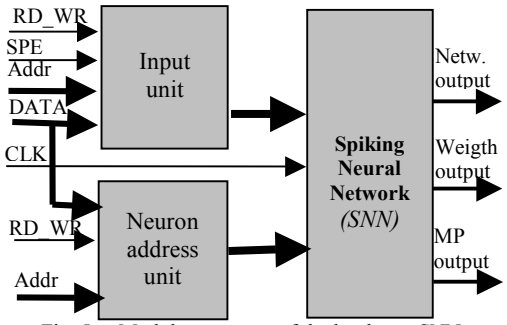

Fig. 5. – Modular strucutre of the hardware SNN

PIII800Mhz PC, a PIV3.2GHz PC for the VHDL design and a Tektronix-TDA 34 channel Logic Analyzer used to perform live measurements on the running network. Still, the I/O signals of the SNN had to be multiplexed using auxiliary modules inside the FPGA (Fig. 5. addressing, data). A VisualC++ application, running on the PIII, delivers inputs and control signals, reads outputs and network

parameters, supervises the learning process The input values (or patterns) are divided into four groups. If the pattern contains more input spikes 1's in the upper half, then it is considered, that the gas-yield is too high, otherwise it is too low. The network has to activate the proper outputs in order to control the process. The experiments performed with the FPGA built SNN were divided into the following phases: 1-delivering the input spikes, 2 - Learning and simulation, 3 - Reading weights, 4 - Reading membrane potentials.

4.2 Description of the supervised learning process

The system was implemented using a composite rule, modified into a supervised Hebb learning algorithm, suitable for the process control application. During the supervised learning phase, the weight values of each synapse of all neurons were testes at each step. Figure 6 shows a measurement when the pressure was too high (MSB bits active in input pattern). By analyzing the data form the first group of synapses of neuron 1 (Fig. 6.A), one can tell, that the weight values are rising,

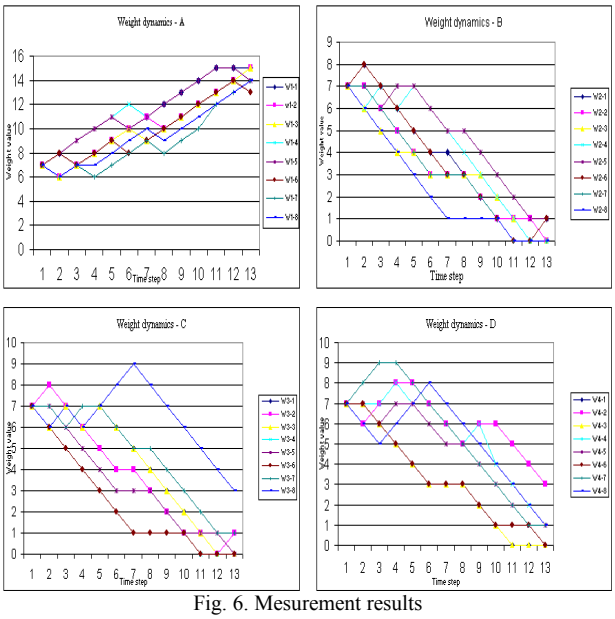

Fig. 6. Mesurement results
The read back weight values show, that one of the neurons has learned to signal too high gas-yield

due to the fact, that both pre- and postsynaptic neurons are active. All other synapses (Fig. 6. BCD) are decreasing their weight values, due to the inactive inputs. To verify if the SNN is learning properly, the MPs were sporadically tested as well, as well (Fig. 7.). The value of the THP was set to 90 in the experiment when data in Fig. 7. has been acquisitioned, the resting potential was set to 10. When the MP of one of the neurons exceeded the threshold value, that neuron emitted a spike, then the MP dropped to the hyperpolarization level of 0.

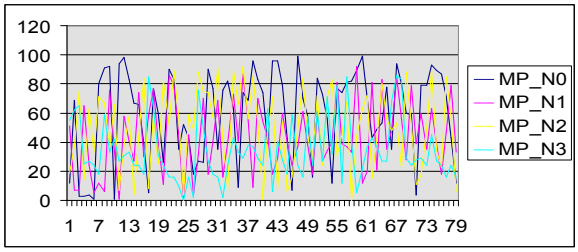

Fig. 7. Measurement results - The read back MP values during the learning process (X – time step, Y – MP value)

4.3 Conclusions

Concluding, it can be said, that the designed model is viable, and its performance ensures real-time process control. The network complexity (32 somas by 32 synapses each) is approximately equivalent to a perceptron model based ANN of ~1024 neurons. Future plans are to adapt the topology, too, by the means of a genetic algorithm and exploiting the partial reconfigurability of the Xilinx FPGAs.

ACKNOWLEDGMENT

The current research is part of the first author's PhD thesis theme and has been partly funded by the Research Institute of the Sapientia Foundation.

References:

[1] László Bakó, Iuliu Székely, Tihamér Sándor Brassai, Development of advanced neural models. Software and hardware implementations, Transactions on Electronics and Communications, Tom 49(63), Fascicola 1-2, Editura Politechnică Timişoara, 2004, ISSN 1583-3380, pp. 214-219

[2] Abeles, M., 1991. Corticonics: Neural Circuits of the Cerebral Cortex. Cambridge University Press.

[3] S. Haykin, Neural Networks, A Comprehensive Foundation, 2nd ed., Prentice-Hall, New Jersey, 1999.

[4] A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve, A. L. Hodgkin and A. F. Huxley, 1952, Journal of Physiology, 117, 500-544.

[5] Gerstner, W., 2001. Spiking neurons In: Maass, W., Bishop, C.M.(Eds), Pulsed Neural Networks. MIT Press, pp.3–53.

[6] Stanton, P. K. and Sejnowski, T. J. Associative long-term depression in the hippocampus induced by hebbian covariance. Nature, 339:215–218., 1989

[7] G. Frank and G. Hartmann, "An artificial neural-network accelerator for pulse-coded model neurons," in Proc. Int. Conf. Neural Networks ICNN95, vol. 4, Perth, Australia, 1995, pp. 2014–2018.

[8] A. Jahnke, T. Schoenauer, U. Roth, K. Mohraz, and H. Klar, "Simulation of spiking neural networks on different hardware platforms," in Proc. ICANN'97. Berlin, Germany, 1997, pp. 1187–1192.

[9] A. Jahnke, U. Roth, and T. Schoenauer, "Digital simulation of spiking neural networks," in Pulsed Neural Networks, W. Maas and C. M. Bishop, Eds. Cambridge, MA: MIT Press, 1998.

[10] ThomasP.Trappenberg, Fundamentals of Computational Neuroscience, Oxford University Press, 2002.