Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 51(65), Fascicola 1, 2006

A Generic Building Block for Hebbian Neural Network with On-Chip Learning

Alin Tisan[1], Ciprian Gavrincea[2], Ştefan Oniga[3]

Abstract – In this paper, we present a digital hardware implementation of an artificial neuron on-chip unsupervised trained with Hebbian rule. The main characteristics of this solution are on-chip learning algorithm implementation and high reconfiguration capability and operation under real time constraints.

Keywords: fpga, learning on-chip, ANN

I. INTRODUCTION

In respond to highly parallelism, modularity and dynamic adaptation, the artificial neural network (ANN) become the most explored data processing algorithms. In addition to this the digital hardware implementation of ANNs in reconfigurable computing architectures like FPGAs circuits, become the easiest and fastest way to reconfigure in order to adapt the weights and topologies of an ANN.

In this paper we present an extendable digital architecture for the implementation of a Hebbian neural network using field programmable gate arrays (FPGAs) and we propose a design methodology that allows the system designer to concentrate on a high level functional specification. For this reason we developed a new library Simulink blocksets constituted by Simulink Xilinx blocks and VHD blocks. With these new created blocks, the designer will be able to develop the entire neural network by parameterize the ANN topologies as number of neurons and layers.

The implementation goal is achieved using the Mathworks' Simulink environment for functional specification and System Generation to generate the VHD code according to the characteristics of the chosen FPGA device.

The design methodology is not new; there have been recent

II. HEBBIAN NEURAL NETWORK

The Hebbian neural network is a multilevel model of perception and learning, in which the 'units of thought' were encoded by 'cell assemblies', each defined by activity reverberating in a set of closed neural pathways The essence of the Hebb synapse is to increase coupling between coactive cells so that they could be linked in growing assemblies. Denoting the neurons by n_i and n_j and the weight that connect the n_j and n_i by w_{ij} and if neuron n_i receives positive input x_j while producing a positive output y_i, the hebbian rule states that for some learning rate $\eta > 0$:

$$w_{ij} := w_{ij} + \Delta w_{ij}, \tag{1}$$

where the increase in the weight connecting n_j and n_i can be given by:

$$\Delta w_{ij} := \eta y_i x_j, \tag{2}$$

Here

$$y_j = f(net) = f\left(\sum_{i=1}^{N} w_{ij} x_i - b\right) \tag{3}$$

where *f(net)* is defined by the discontinuous threshold activation function *sgn(net)*:

$$\operatorname{sgn}(net) = \begin{cases} 1 \text{ if } net \geq 0 \\ -1 \text{ if } net < 0 \end{cases} \tag{4}$$

Of all the learning rules, Hebbian learning is probably the best known. It established the foundation upon which many other learning rules are based. For this reason, we developed this learning rule first.

Hebb proposed a principle, not an algorithm, so there are some additional details that must be provided in order to make this computable:

- It is implicitly assumed that all weights w_{ij} have been initialized (e.g. to some small random values) prior to the start of the learning process.
- The parameter η must be specified precisely (it is typically given as a constant, but it could be a variable).

[1]Universitatea de Nord din Baia Mare, Facultatea de Inginerie, Departamentul de Electrotehnică, Str. V. Babeş, nr. 62A 430083 Baia Mare, e-mail atisan@ubm.ro
[2] Universitatea de Nord din Baia Mare, Facultatea de Inginerie, Departamentul de Electrotehnică, Str. V. Babeş, nr. 62A 430083 Baia Mare, e-mail gcg@ubm.ro
[3] Universitatea de Nord din Baia Mare, Facultatea de Inginerie, Departamentul de Electrotehnică, Str. V. Babeş, nr. 62A 430083 Baia Mare, e-mail onigas@ubm.ro

- There must be some type of normalization associated with the increase of the weight or else w_{ij} can become infinite;
- Positive inputs tend to excite the neuron while negative inputs tend to inhibit the neuron.

III. BLOCKSET NEURAL NETWORK DESIGN

In order to learn on-chip, the Mc Culloc - Pitts neuron model, i.e. each of the input vector components xi is multiplied with the corresponding weight w_{ij}, and these products are summed up yielding the net linear output, upon which the threshold activation function is applied to obtain the activation which is either 1 or −1, was modified to make the calculate the weights according to a certain learning rule and to update the new weights into a weight memory block, figure 1.

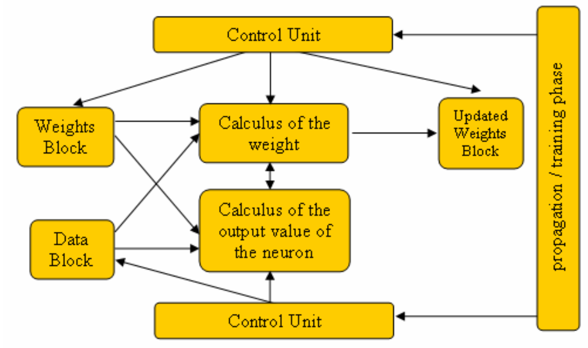


Fig.1. Block level representation of the neuron with on-chip learning

The parallelism adopted is a node parallelism one and requires one multiplier per neuron, therefore all neurons will work in parallel. If data inputs are memorized in a single memory block, the weights storage will be private for each neuron because all the neurons have to access their correspondent weight memories at the same time.

The proposed model of the neuron is constituted by two major blocks: a control logic bloc and a processing block.

The control logic block will manage the control signal of the processing bloc in order to initialize and command the processing components.

The processing block is design to calculate the neural output, the weights according to learning rule adopted, in this case the Hebbian rule, and to update these weights.

A. *Control logic block*

The control logic block is described in VHDL code and is incorporated into design by a black box HDL, figure 2.

The role of this bloc is to load from Mathlab workspace the following variables: the number of vectors used for training and the number of bits used for data representation.

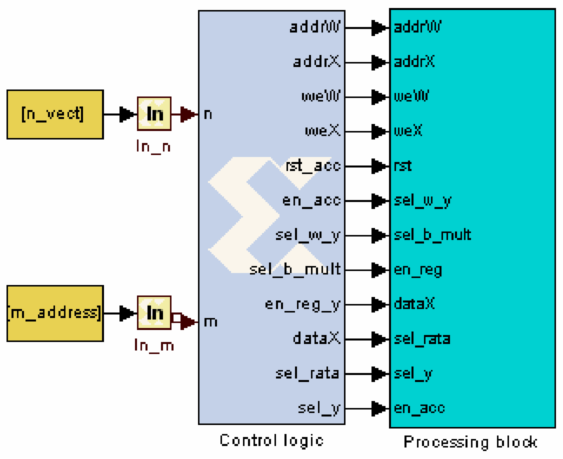
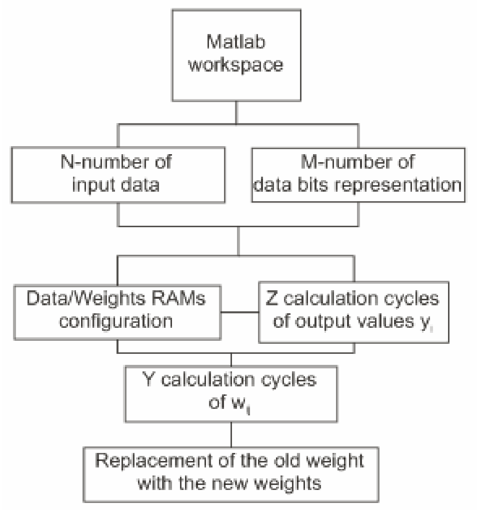


Fig. 2. Blockset architecture of neuron

Depending on these variables, the control logic block will configure the size of the RAMs used for data and weights storage and will manage the enable signals of the processing elements of the processing block in order to run the processing block in a propagation phase or in a training phase, figure 3..


Matlab workspace
N-number of input data
M-number of data bits representation
Data/Weights RAMs configuration
Z calculation cycles of output values y
Y calculation cycles of w
Replacement of the old weight with the new weights
Fig.3. Block level representation of the neuron with on-chip learning

The enabling algorithm of processing elements depends on the number of input neurons, the size of the block memories that storage the data or weights vectors and the delays introduced by the different processing or storage elements.

B. *Processing Block*

The processing block is the main block of the design. It incorporates both the artificial neuron and the logic for on-chip learning algorithm.

The structure of the artificial neuron consist in two memory blocks, one for data samples and one for weight coefficients, and one MAC unit, figure 4.

The logic for the learning algorithm requires a MAC unit too, but in order to save hardware resources we decide to use the MAC unit of the artificial neuron for the implementation of the learning algorithm. To achieve this, the design requires a number of multiplexer blocks and special control logic.

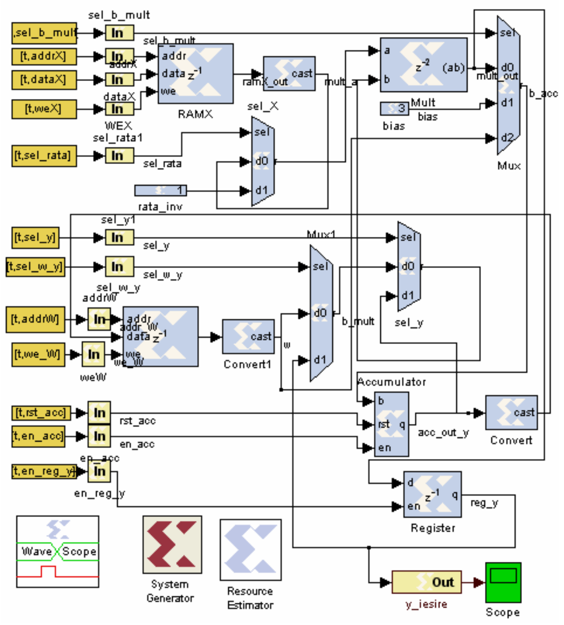


Fig. 4. Architecture of the processing block.

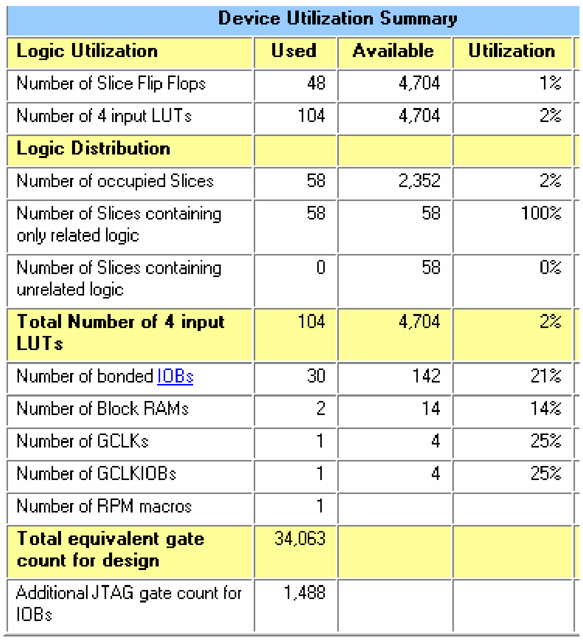
Artificial neuron with on-chip learning has two modes of operation: propagation mode on which the design acts as a regular artificial neuron and learning mode. On learning mode there are two stages: on the first stage an output of the artificial neuron is calculated based on the data provided by training vector and on the second stage weight coefficients are recalculated based on Hebbian learning rule.

By using one MAC unit it saves hardware resources but execution time slightly increases. This happens only on the learning mode and doesn't affect the propagation mode. For an artificial neural network time constrains are important only on propagation mode of operation, and having a longer learning period doesn't affect the performance of the artificial neural network.

IV. HARDAWARE IMPLEMENTATION

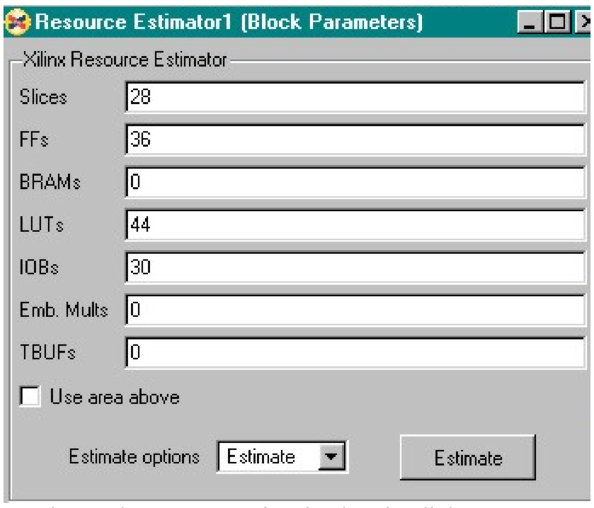
The design is implemented into Digilab 2E (D2SB) development board featuring the Xilinx Spartan 2E XC2S200EPQ208-6 FPGA. This chip has 2352 slices (control unit which includes two 4-inputs look-up tables (LUT) and two flip-flops) and 14 block RAMs. The resources usage of a single neuron were estimated by ISE Xilinx and by Simulink Resource Estimator Block and are shown in fig. 5 and fig. 6

The differences between these two estimators come up because of different way of resource usage calculation of the logic blocks implementation in FPGA



Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	48	4,704	1%
Number of 4 input LUTs	104	4,704	2%
Logic Distribution			
Number of occupied Slices	58	2,352	2%
Number of Slices containing only related logic	58	58	100%
Number of Slices containing unrelated logic	0	58	0%
Total Number of 4 input LUTs	104	4,704	2%
Number of bonded IOBs	30	142	21%
Number of Block RAMs	2	14	14%
Number of GCLKs	1	4	25%
Number of GCLKIOBs	1	4	25%
Number of RPM macros	1		
Total equivalent gate count for design	34,063		
Additional JTAG gate count for IOBs	1,488		

Fig. 5. The resource estimation by ISE Xilinx of a single neuron



Resource Estimator1 (Block Parameters)

Xilinx Resource Estimator

Slices	28
FFs	36
BRAMs	0
LUTs	44
IOBs	30
Emb. Mults	0
TBUFs	0

☐ Use area above

Estimate options: Estimate

Fig. 6. The resource estimation by Simulink Resource Estimator of a single neuron

Because Spartan 2E doesn't have implemented dedicated MAC units, we designed a multiply and accumulate structure with Xilinx blocks of Simulink Xilinx Blockset library, figure 7.

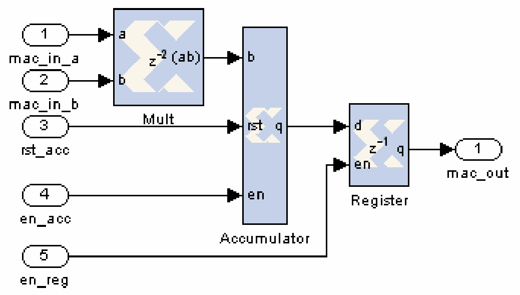


Fig. 7. MAC structure designed with Xilinx blocks

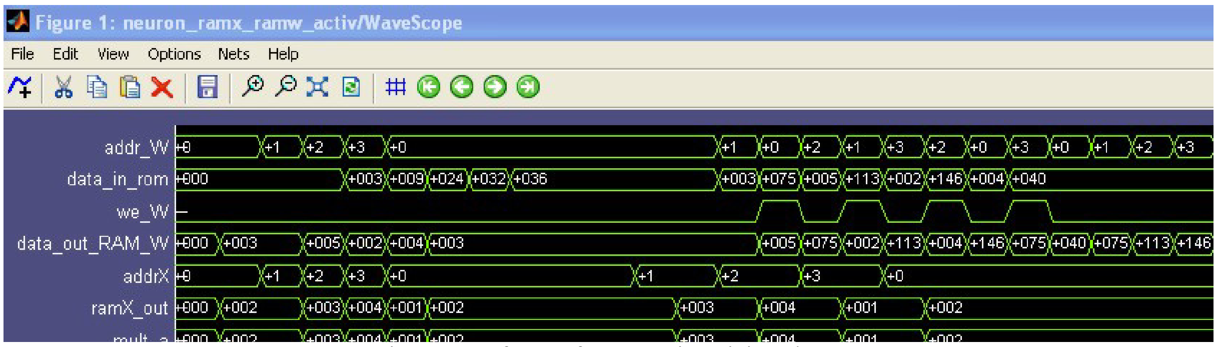


Figure 1: neuron_ramx_ramw_activ/WaveScope
File Edit View Options Nets Help
addr_W
data_in_rom
we_W
data_out_RAM_W
addrX
ramX_out
mult_a

Fig. 8. Waveforms of a neuron in training phase

Because, multiplication block use the largest resources, 12 slices, and in order to implement the three multiplications needed to calculate the updated weights and the neuron output with one multiplication and accumulation block is necessary to add three more multiplexers block to select the right signal to add or to multiply.

The total delay is gave by the number of training vectors plus 3 other cycles (1 for RAM, two for the multiplier) for neuron output calculation and 2 x number of training vectors plus 12 cycles to calculate the new weights and to update the weight RAM.

The total number of cycles needed to calculate and update the weights is presented in fig. 8.The waveforms confirm the calculation algorithm, for a given number of training vectors, of total delay for a neuron in training phase and propagation phase

V. CONCLUSION

We have presented hardware architecture of artificial neuron with on-chip learning controlled by a generic control unit described in VHDL code. This method uses minimal hardware resources for implementation of this kind of artificial neuron. The main advantage of this solution is highly modularity and versatility in neural network designing.

In order to design and to implement the neuron we used the Mathworks' Simulink environment for functional specification, System Generation to generate the VHD code according to the characteristics of the chosen FPGA device and ISE Xilinx to simulate the design at different stages of implementation and to generate the bit file.

The neuron designed is a generic module and can be used to design neural networks that have the following features:

- the training is on-line;
- the learning is on-chip;
- all weights have been initialized prior to the start of the learning process;
- the learning parameter must be specified precisely;
- there must be some type of normalization associated with the increase of the weight or else w_{ij} can become infinite;
- positive inputs will tend to excite the neuron while negative inputs will tend to inhibit the neuron;
- the initialization of the data and weigh RAMs must be done through m file from Matlab environment.

These results will be used to design other learning rules starting from a Hebbian one, and also to make the ANN design more modularized so that its size can be modified as randomly increased or decreased of the neuron number in order to find out that ANN which fits to a specific application. In this way the FPGA hardware implementation makes ANN more convenient to be carrying, modularized and reconfigurable.

REFERENCES

[1] E. Fiesler, R Beale, "Handbook of neural network", Oxford University Press, 1997.
[2] A. Singh, "Design & Implementation of Neural Hardware". University School of Information Technology, GGS Indraprastha University, Delhi http://www.geocities. com /aps_ipu/papers/ synopsis.pdf, 2005.
[3] A., Bernatzki, W, Eppler, "Interpretation of Neural Networks for Classification Tasks." Proceedings of EUFIT 1996, Aachen, Germany, http://fuzzy.fzk.de/eppler /postscript/eufit.ps.2005
[4] A. Savran, S. Unsal. Hardware Implementation of a Feedforward Neural Network using FPGAs. International Conference on Electrical and Electronics Engineering. Bursa, December 2003.
[5] Yihua Liao. Neural Networks in Hardware: A Survey. Department of Computer Science, University of California, Davis.
[6] J. Zhu, P. Sutton. FPGA Implementations of Neural Networks – a Survey of a Decade of Progress. Proceedings of 13th International Conference on Field Programmable Logic and Applications (FPL 2003), Lisbon, Sep 2003.
[7] S. Oniga, Hand Gesture Recognition System Using Artificial Neural Networks implemented in FPGA, PhD Thesis, 2006.