Hardware Implementation of Simple Competitive Artificial Neural Networks with Neuron Parallelism

Article · January 2007

CITATIONS
3

READS
90

4 authors:

Stefan Oniga
Universitatea Tehnica Cluj-Napoca
82 PUBLICATIONS **681** CITATIONS
SEE PROFILE

Alin Tisan
Royal Holloway, University of London
36 PUBLICATIONS **694** CITATIONS
SEE PROFILE

Buchman Attila
Universitatea de Nord din Baia Mare
12 PUBLICATIONS **79** CITATIONS
SEE PROFILE

C. Lung
Technical University of Cluj Napoca North University Center of Baia Mare
52 PUBLICATIONS **208** CITATIONS
SEE PROFILE

Hardware Implementation of Simple Competitive Artificial Neural Networks with Neuron Parallelism

Ştefan Oniga, Alin Tisan, Attila Buchman, Claudiu Lung

North University, Baia Mare, Electronic and Computer Engineering Department
Dr. V. Babes Street 62A, Baia Mare, 430083, Romania
Phone: (40)-(362)-401-265, Fax: (40)-(262)-276-153, onigas@ubm.ro

Abstract

Competitive self-organizing and self learning neural networks, also known as self-organizing feature maps (SOFM), represent one of the most interesting types of the artificial neural networks (ANN). This paper presents the successful implementation of some simple competitive neural networks with neuron parallelism used in model classification tasks in field programmable gate arrays (FPGA). The network design was carried out using the System Generator software, which is also used to generate the VHDL code for the network. Xilinx ISE 8.2i was used for synthesis and implementation.

1. INTRODUCTION

A competitive ANN with neuron parallelism consists of a control block, an input layer and an output layer. The output layer consists of a number of neurons equal to the number of classes in which we would like to classify and an activation function calculus block, common to all the neurons. The neuron consists of a weights memory specific to each neuron and a calculus block used to calculate the simplified Euclidian distance. The following figure presents a competitive ANN with neuron parallelism, having the same parameters as the ANN implemented with the layer parallelism. (7 neurons in the input layer and 15 neurons in the output layer).

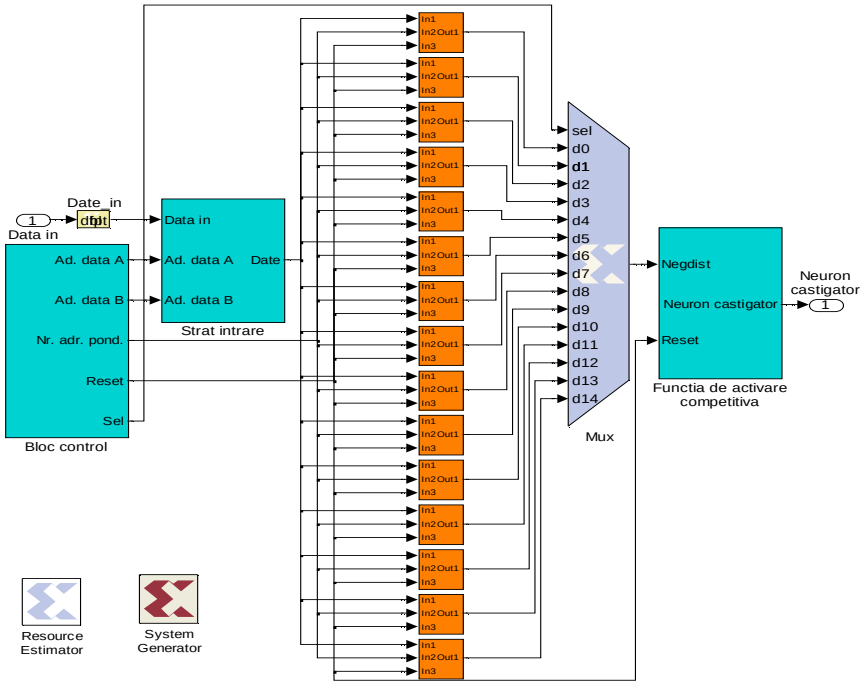


Fig. 1 The hardware model of the competitive ANN with neuron parallelism

2. IMPLEMENTING A COMPETITIVE ANN WITH LAYER PARALLELISM

The changes in the structure of the ANN in order to implement the neuron parallelism will be presented next. The command signals are being generated by the command block that has to be adapted function of the number of neurons. This block generates the address where the input vector will be entered into the input layer, the block being designed in the form of a dual port memory. The command block generates the address form which the date of the input layer are read, data used afterwards also by the weight memories from within the structure of each neuron. It also generates reset signals and the selection signal for the multiplexer. Adapting the command block is easy because the parameters regarding the number of neurons, the numbers of operations during an input signal sample, the delays, etc., are being automatically loaded from the Matlab IDE. The signals generated by the command block are presented in figure 2.

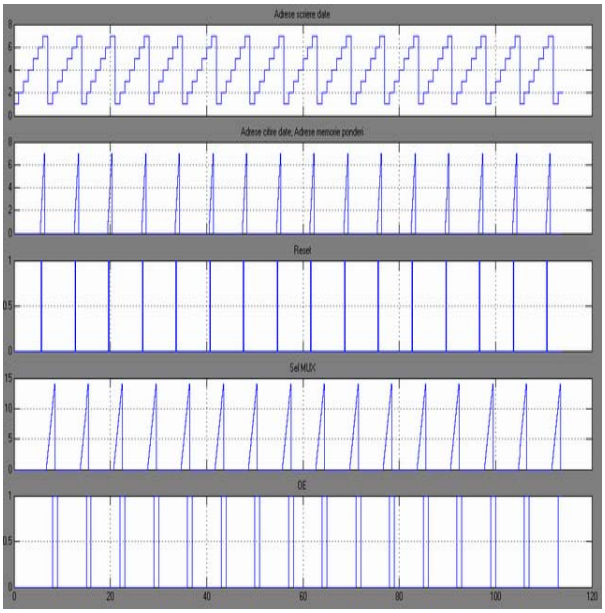


Fig. 2 The signals generated by the command block

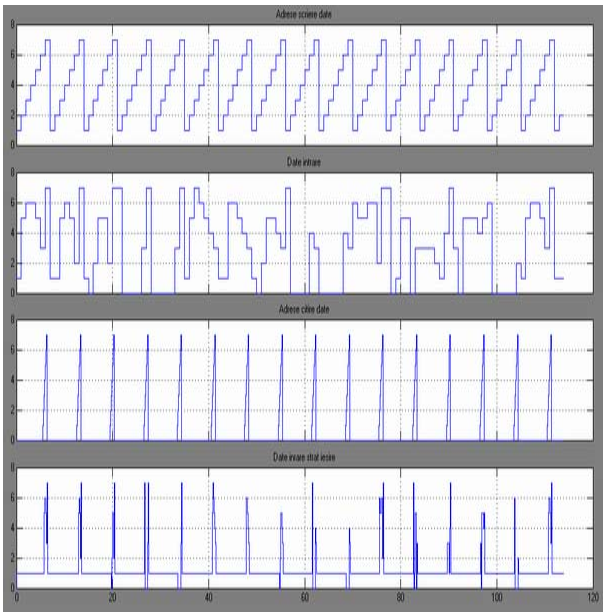
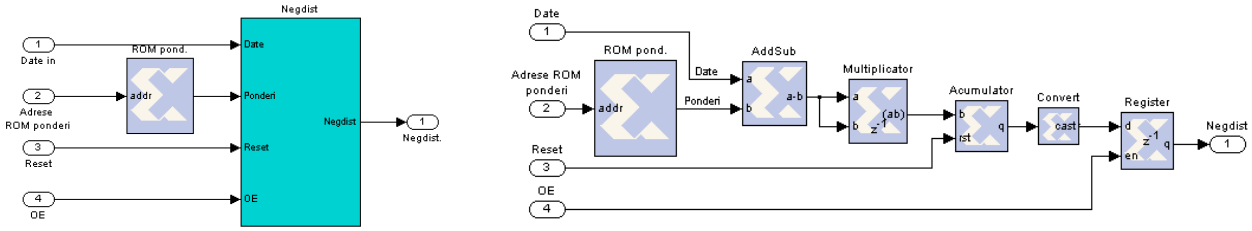


Fig. 3 Input layer wave forms

The input layer has the same structure as in the case of the ANN with layer parallelism, the only difference being the frequency with which data is being entered and read. Fig. 3 present the waveforms corresponding to the input layer.

A neuron contains, besides the block that calculates the Euclidian distance, a weights memory, as presented in the next figure:



a) Block diagram

b) The structure based on Xilinx blocks

Fig. 4 The neuron within the structure of a competitive ANN with neuron parallelism

In Fig. 4.b the structure of a neuron created using Xilinx blocks is being presented. The AddSub block calculates the differences between the data and the weights, while the multiplication block calculates the square of these differences,

the accumulator block sums up the squares of the differences, the results being stored temporally in the output register.

In order to calculate the smallest value and determine the neuron which has the minimal output, a block that calculates the competitive function activation has been implemented, block similar to the one in Fig. 6.6. Because there is only one such block, the neuron outputs are fed in sequence to its input by the means of a 15 input multiplexer. The selection signal for the multiplexer is generated by the command block.

A detailed view with the waveforms corresponding to the first three test vectors for the competitive ANN with neuron parallelism are presented in Fig. 5.

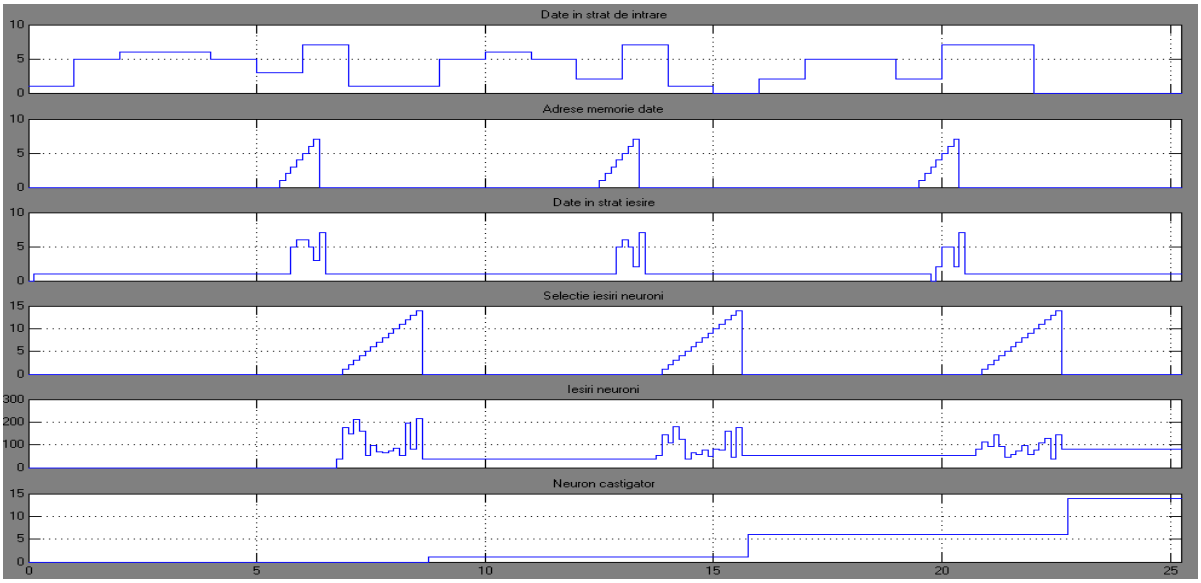


Fig. 5 The waveform corresponding to the answer of the ANN to the first three vectors

3. RESULTS

The results of the implementation of the competitive ANN with neuron parallelism in a FPGA XC2V1000 device are presented in the table 1.

The resources used in the implementation of a neuron are only 12 slices, 1 RAM block and one dedicated multiplier. The total of resources necessary to implement the competitive ANN with 15 neurons represents only a small percentage of the total resources available in the circuit, except the memory blocks and the dedicated multipliers which are used in 40 and respectively 37.5 percent. In order to implement more than 40 neurons, the multiplying block can be implemented using distributed logic instead of the dedicated multipliers, their number being limited to 40.

TABLE 1
THE RESOURCES UTILIZED BY THE COMPETITIVE ANN WITH NEURON PARALLELISM

	Command block	Input Layer	15 Neurons (res/neuron)	Neuron outputs multiplexer	Activation function	TOTAL	% of XC2V1000
Slices	19	0	12	43	29	271	5,27
Flip-flops	17	0	18	2	23	312	3,05
RAM blocks	0	1	1	0	0	16	40
LUTs	29	0	12	81	35	325	3,18
Multipliers	0	0	1	0	0	15	37,5

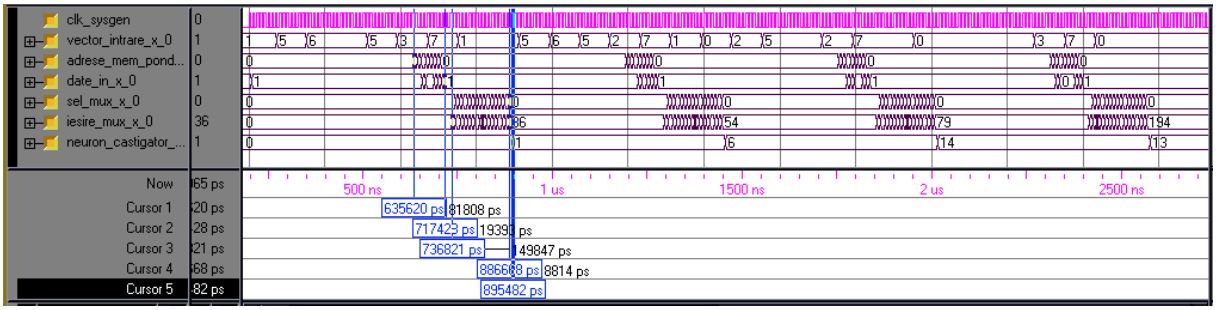
In this case, the resources per neuron increase to 22 slices, 30 flip-flops, 1 RAM block, 31 LTUs, 0 multipliers. Also, such memories can be implemented using the distributed memory available in the device (160 Kb) instead of using the block RAM memories. The necessary resources increase in this case with 3 slices, 3 flip-flops and 3 LUTs. We can estimate the maximum number of competitive neurons that can be implemented into this device, as in the following table:

TABLE 2
RESOURCES UTILISED BY THE MAXIMUM NUMBER OF NEURONS THAT CAN BE IMPLEMENTED IN THE XC2V1000

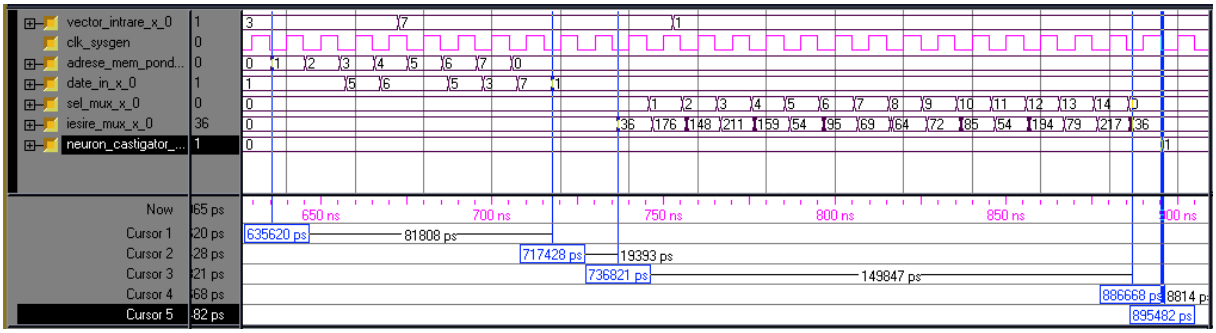
	ANN 15 Neurons, ver. 1	ANN 40 Neurons ver. 1	Total XC2V1000	Available	ANN 15 Neurons ver. 2	No. of neurons	TOTAL : 40 neurons, ver. 1 + 141 neurons, ver. 2
Slices	271	723	5120	4397	466	141	5103
Flip-flops	312	832	10240	9408	640	221	6848
RAM blocks	16	40	40	0	0	-	40
LUTs	325	867	10240	9373	655	215	7024
Multipliers	15	40	40	0	0	-	40

From the calculations we can estimate that the maximum number of neurons, with the corresponding circuitry, that can be implemented in the device chosen is around 181. Of these, 40 will be implemented with dedicated multipliers and 141 using distributed logic.

The results of the functional simulation of the implemented network, more precisely a detail on the first 4 test vectors and over the computational time intervals for the answer for the first test vector are presented in Fig. 6.a and respectively 6.b.



a) Detail of the first 4 vectors



b) Detail on the computation times for the first test vector

Fig. 6 Functional simulation of the competitive ANN with neuron parallelism

The simulation ran with a 100MHz clock signal. The elements of the input vector are written in the memory of the input layer in a sequential manner, at equal time intervals.

$$T_s = (n_1+1)T_{clk} = 80 \text{ ns} \quad (1)$$

The duration of the input vector is 560 ns.

The input layer transmits in sequence these elements, plus the bias, to the output layer, at time intervals of Ts/(n1+1). Each of the n2 neurons of this layer makes (n1+1) computations. The transfer of the outputs to the block that computes the competitive function activation is done at 15 clock intervals. The neuron that wins the competition is determined after 8.8 ns. The results of the simulation show that the first test vector from the figure [1 5 6 6 5 3 7] is assigned the the first class, the second vector [1 1 5 6 5 2 7] to the 6th class, and so on. The maximum frequency of the clock signal resulted from the synthesis report is 123.7 MHz. The maximum frequency with which the input vector elements can be applied is:

$$F_{s\,max} = F_{clk\,max}/(n_1+1) = 15{,}45 \text{ MHz} \quad (2)$$

4. CONCLUSIONS

This paper presented the successful implementation of some simple competitive neural networks used in model classification tasks.

The implemented network is of neuron parallelism type, having as many Euclidian computation blocks as neurons are within the network, and only one function activation block. The structure of the network modifies function of the number of neurons that have to be implemented. This network also correctly classifies all the training and test vectors supplied. The resources used depend on the number of neurons. In the case of the 15 neurons network, it utilizes: 5.27% of the total number of slices and 37.5% of the total number of dedicated multipliers. The maximum number of competitive neurons that can be implemented into a device such is the one specified above is estimated to be around 181. The maximum frequency at which the vectors can be applied at the input of the neural network is 15.45 MHz.

The model is to be developed function of the number of neurons, and the size of the network should not exceed 180 neurons for the specified circuit.

For larger networks and higher working frequencies, lager FPGA devices with higher working frequencies can be used. For example, the XC2VP125 of the Virtex II Pro family contains over 556 dedicated multiplexers and over 55000 slices, which would allow the implementation of over 2000 competitive neurons. The working frequency of these devices can go as high as 400-500 MHz. The frequency of the input signal is $1/(n_1+1)$ of the maximum frequency of the device, where n1 represent the number of neurons of the input layer (the number of sizes of the input vector).

Among the author's contributions to this chapter we can mention:

- Hardware design of the Negdist block which allows calculating the sum of the squares of the subtraction operation between the elements of two vectors.
- The development of an algorithm used to determine the neuron for which the distance between the weight vector and the input vector is minimal.
- Conception of the hardware model for the competitive function activation block.
- Modelling of the competitive ANN with neuron parallelism in Simulink/System Generator.

The estimation of the resources used from within the FPGA device and the selection of the FPGA which has the most suitable characteristics.

- Finding of the maximum input signal frequency function of the maximum frequency of the network and the parameters of the network (the number of inputs and respectively the number of neurons).
- Design of a block used to evaluate the errors of the hardware model.
- The simulation, implementation and experimental verification of all the models designed using the hardware platform.

REFERENCES

[1] J. Starzyk, Y. Guo. "A Self-Organizing Learning Array and its Hardware-Software Co-Simulation", Proc. ECCTD, Krakow, Poland, 2003

[2] T. Kohonen. Self-Organizing Maps. Third Edition. Springer-Verlag Berlin, 2001

[3] S. Oniga, "A New Method for FPGA Implementation of Artificial Neural Network Used in Smart Devices", International Computer Science Conference microCAD 2005, Miskolc, Hungary, March 2005, pp. 31-36

[4] A. Tisan, S. Oniga, A. Buchman, C. Gavrincea, Architecture and Algorithms for Syntetizable Neural Networks with On-Chip Learning, International Symposium on Signals, Circuits and Systems, ISSCS 2007, July 12-13, 2007, Iasi, Romania, vol.1, p. 265 - 268, ISBN 1-4244-0968-3, IEEE Catalog Number: 07EX1678, Library of Congress: 2007920356

[5] S. Oniga, A. Tisan, D. Mic, A. Buchman, A. Vida-Ratiu, Hand Postures Recognition System Using Artificial Neural Networks Implemented in FPGA, 30th International Spring Seminar on Electronics Technology, ISSE 2007. Technical University of Cluj-Napoca, ROMANIA, May 9-13, 2007, p. 507 - 512, ISBN 1-4244-1218-8, IEEE Catalog Number: 07EX1780C, Library of Congress: 2007924573