Hardware implementation of simple competitive neural networks with layer parallelism

Ştefan Oniga, Alin Tisan, Daniel Mic, Attila Buchman, Ciprian Gavrincea and Andrei Vida

Electrotehnical Department, North University, Baia Mare, Romania
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 layer 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

Competitive self-organizing and self learning neural networks, also known as self-organizing feature maps (SOFM), represent one of the most interesting types of ANN. The way information is organized in the human brain inspired these types of ANN.

The task of the competitive networks is to classify input models. Similar models are classified into the same class, represented by the same output unit. Each neuron will become specialised in recognising certain features of the input data.

2. THE STRUCTURE

Self organising neural networks are characterised by the fact that they actually learn unsupervised to discover features, regular patterns and correlations of the input data. The neurons of the simple competitive networks are arranged in a one-dimensional output layer, totally connected to the neurons of the input layer through some stimulating weights. They are permanently in a competitive state, at a certain point in time only one being active.

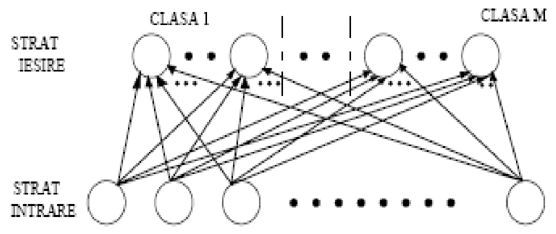


Figure 1 ANN with simple competitive learning

Each neuron has as many input connections as the number of attributes used in classifying. Starting with a set of initials weights, randomly generated, the training procedure consists in finding the neuron that has the closest weights compared to the input vector and declaring that neuron as winner. There are two methods of determining similarity. The first determines the net input of each neuron according to the equation:

$$net_k = \sum_i^N w_{ki} x_i \tag{1}$$

The neuron for which the net input has the highest value is declared the winner.

The second method calculates the distance between the input vector and the weights vector, the most commonly used being the Euclidian distance, as in the following equation:

$$net_k = \sqrt{\sum_i^N \left[x_i - w_{ki}\right]^2} \tag{2}$$

The neuron for which the equation 2 has the smallest value will win the competition.

The learning process consists of changing the weights according to the equation:

$$\Delta w_{ji} = \eta(x_j - w_{ji}) \quad (3)$$

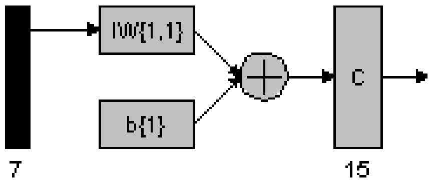
For the j neuron which won the competition, and

$$\Delta w_{ki} = 0, \quad (4)$$

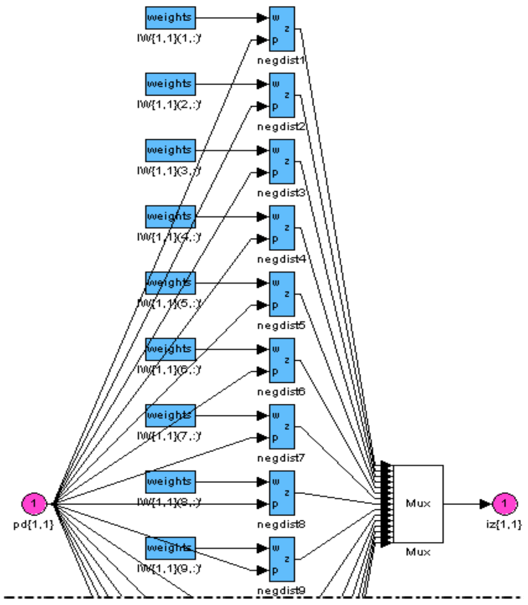
for the neuron k $\neq$ j. η represents the learning rate.

Through this, the weights vector of the winning neuron, j, gets closer to the pattern present at the input.

Upon the completion of the training, the classification process consists of calculating the distance between the input vector and each neuron and declaring the input as pertaining to the class represented by the winning neuron.



a) The ANN Architecture



b) The structure of the IW{1,1} block

Figure 2 Competitive ANN created using NN Toolbox

3. Implementing a simple competitive ANN

The training phase of the ANN was software implemented, while the propagation phase was implemented hardware. We will present below the way the network was trained and subsequently implemented hardware in the propagation phase.

3.1. Training a simple competitive ANN

In order to train a competitive ANN, a simple ANN was created by using Matlab code or by using the graphical interface Network/Data Manager. In the following figure, a competitive network will be presented, composed of 7 neurons in the input layer and 15 neurons in the output layer.

The IW{1,1} block presented in figure 2.b calculates the negative of the Euclidian distance between the applied input vector and the vectors formed by the lines of the weights matrix. If all the biases are zero, the maximum net input of the neuron can be zero, when the input vector is identical to the weights vector of the neuron. The competitive transfer function receives the net input of the neuron and supplies a value of zero for all the neurons, except the winner, that is the one that has the most positive net input, for which is supplies a value of 1.

After training the network with a set of 10x15 test vectors and simulating it, we get the results presented next:

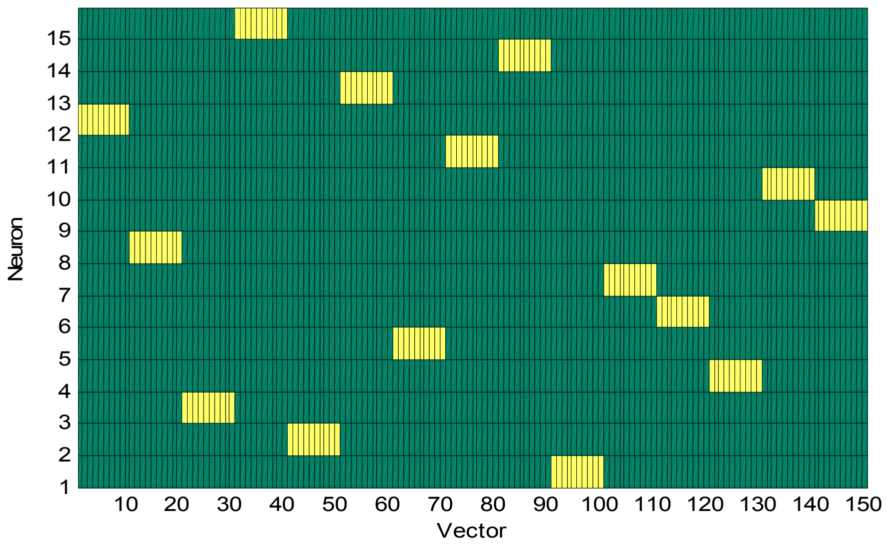


Figure 3 The results of the competitive ANN simulation

It is noticeable that, at a certain point in time, only one neuron is active and that the 15 sets of 10 vectors are each divided into 15 different classes. The first set of 10 vectors is part of class 12, the second set of class 8, and so on.

The results of the simulation being as expected (the classification in 15 different classes of the vector sets) the weights of the network are saved at this point.

3.2 The hardware implementation of the competitive ANN

In order to implement a competitive ANN, a model similar to the one presented in figure 2 was created, using Xilinx blocks. Calculating the Euclidian distance using equation 2 is difficult because of the square root operation that is hard to implement hardware. Due to the fact that the comparison between the neurons can also be achieved by using the square of the distances between the input vectors and the weights of the neurons, implementing the square root calculation is not necessary, so the equation 2 can be replaced by:

$$y_k = \sum_{i}^{N} \left[x_i - w_{ki} \right]^2 \qquad (5)$$

To achieve this, the structure of the neuron will contain a memory for the weights, a block used to calculate the xi-wki subtractions, a MAC block used to calculate the sum of the squares of these subtractions and, finally, the block of the competitive activation function, as presented in figure 4. The activation function block is common for all the neurons in the output layer.

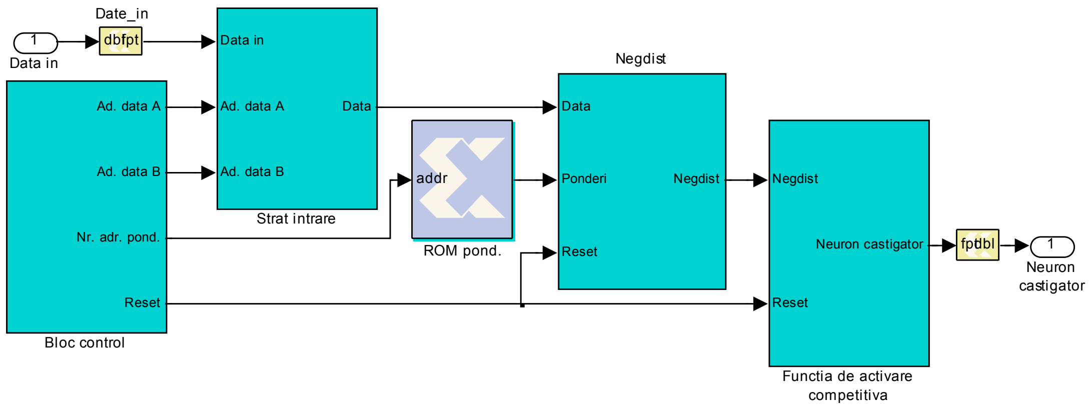


Figure 4 The hardware model of the competitive ANN

The Negdist block consists of an AddSub Block, part of the Xilinx Blockset and a MAC block created by the author. The Addsub block is used to calculate de x_i-w_{ki} subtractions, while the MAC block determines the sum of the squares of these subtractions.

The activation function block, presented next, is composed of two parts. A first part determines the minimum of the y_k type of inputs of the neurons that form the output layer, and the second part determines the position of the neuron which has the minimal output.

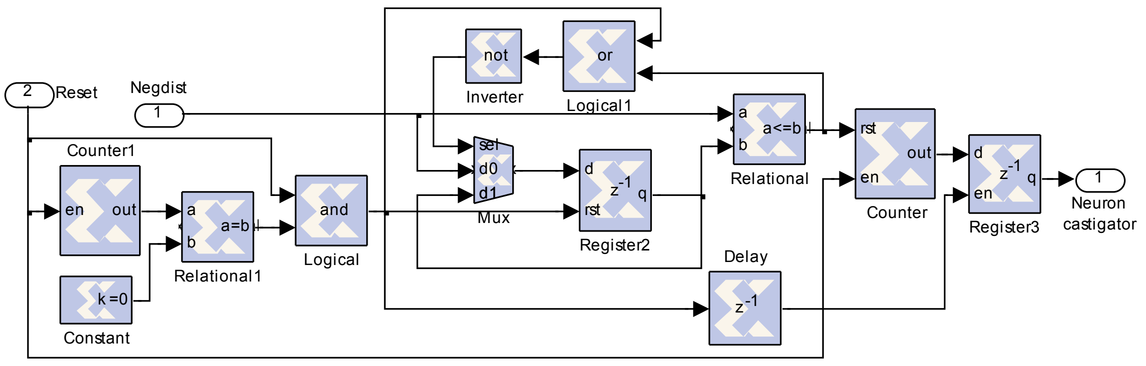
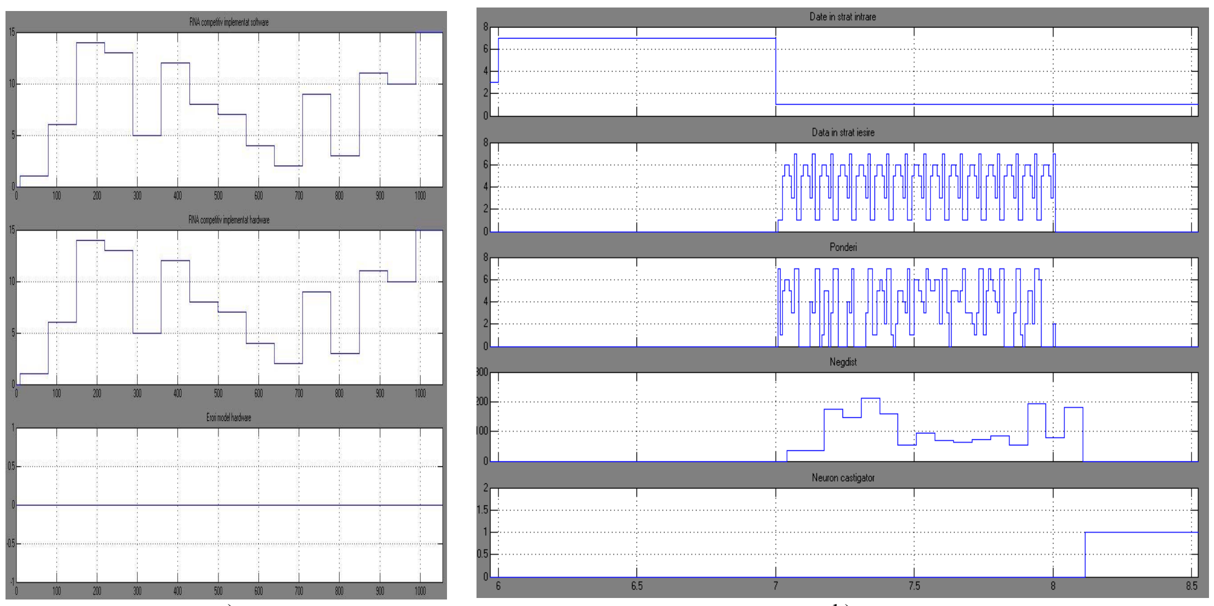


Figure 5 The competitive function activation block

4. Implementing a competitive ANN with layer parallelism

In order to implement a competitive ANN with layer parallelism, a single processing element per layer is required. To verify the performance of a competitive ANN with layer parallelism, a network with an n_1=7 neurons input layer and an n_2=15 neurons output layer was implemented. The hardware model has the same structure as the model of the neuron presented in figure 4. The competitive ANN with layer parallelism consists of a control block, an input layer made with a memory of type Block RAM, a weights memory common to all the neurons, the simplified Euclidian distance calculus block and the competitive function activation block, also shared by all the neurons. The architecture is the same, no matter how many neurons are there on the input, respectively on the output layer. The only differences are the parameters of the blocks, the capacity of the memories and the computational speed.

The results of the network simulation are presented next.



a) b)

Figure 6 Software and hardware simulation of the competitive ANN model

The first graphic, in figure 6.a represents the classification in the 15 classes of the 15x10 input vectors, classification resulting from the software implementation of the network. The second graphic presents the classification made by the hardware model, while the last presents its errors compared to the software model. In figure 6.b, a detailed view of the hardware model simulation is presented

Following the application of a vector at the input of the input layer, this is applied to the output layer at a frequency of $(n_1+1)n_2$ times higher due to the fact that the negdist block must execute n_1+1 computations for each of the n_2 neurons.

As we can see, there is no difference between the classification made by the hardware model compared to the one made by the software implementation. Simulating the hardware model using another set of 15x10 test vectors creates the same results, without any errors.

The resources used by the ANN implemented into a Virtex II XC2V1000 device are presented next:

	Comand block	Input layer	Weights Memory	Dist. calc. block	Activation function	TOTAL
Slices	16	3	0	15	34	68
Flip-flops	15	4	3	25	27	74
RAM blocks	0	1	1	0	0	2
Mem. tables	24	0	0	12	40	76
Multipliers	0	0	0	1	0	1

Table 1 Resources used by the competitive ANN

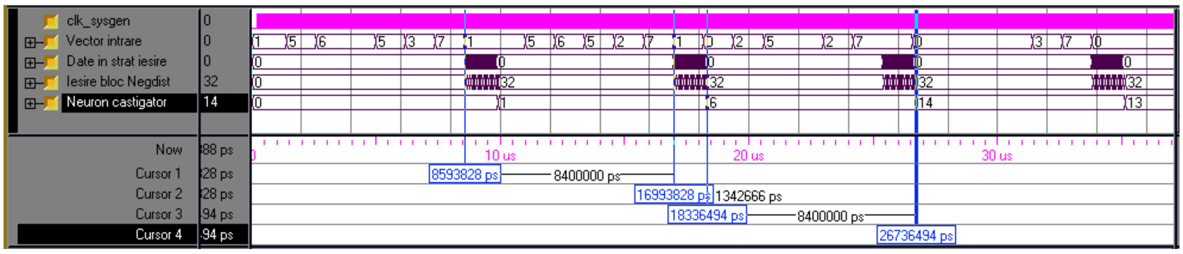
The results of the post-implementation simulation, presented in the next figure, confirm the correct functioning of the implemented circuit. The elements of the input vector are fed to the input layer in a sequential manner with a frequency of 1/Ts,

where Ts represents the duration of a vector element equal to:

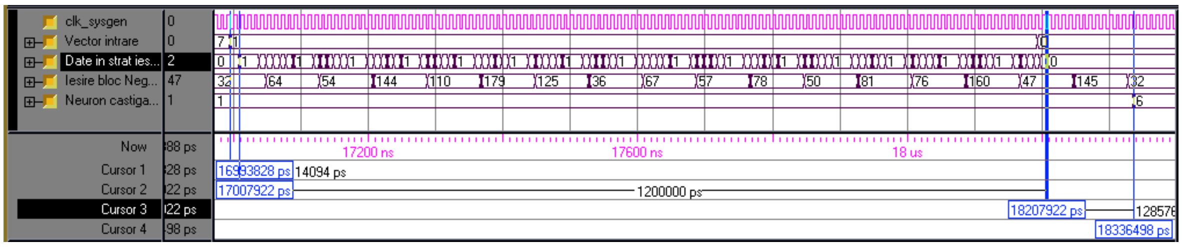
$$T_s = (n_1+1)n_2T_{clk} = 1200 \text{ ns} \qquad (6)$$

for a clock signal frequency of 100 MHz. The duration of the input vector is 8400 ns. The first test vector from the figure ([1 5 6 6 5 3 7]) is assigned to the first class, the winning neuron, after applying the second vector ([1 1 5 6 5 2 7]) is neuron 6, and so on.

The input layer successively transmits these elements to the output layer with a frequency of (n1+1)n2/Ts, allowing it to perform the (n1+1)n2 calculations. The neuron that wins the competition is determined after 128 ns.



a) Detail of the first 4 vectors



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

Figure 7 The post implementation simulation of the ANN with layer parallelism

The maximum frequency of the clock signal resulted from the synthesis report is 136 MHz. The maximum frequency with which the input vector elements can be applied is:

$$F_{s\,max} = F_{clk\,max}/(n_1+1)n_2 = 1{,}133 \text{ MHz} \qquad (7)$$

5. Conclusions

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

The competitive network implemented is the one using layer parallelism, having a single processing element. The ANN implemented classifies correctly all the training vectors and two different sets of 15x10 test vectors. The resources used are minimal, (around 1.3% of a Virtex II XC2V1000 device for a network of 15 neurons) fact that permits the development of large size networks. The maximum frequency of the clock signal is 1.133 MHz.

This type of network presents an advantage for the cases in which the frequency of the patterns that must be identified is under 1MHz, because it allows the development of large networks without modifying the structure of the circuit. If the input vectors have a frequency between 1MHz and 15MHz, a network with neuronal parallelism must be used.

Among the author's contributions 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 (equation 5).
- 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.
- The modelling of a competitive ANN with layer parallelism using the IP blocks.
- 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).

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