SADIO Electronic Journal of Informatics and Operations Research

http://www.dc.uba.ar/sadio/ejs

vol. 6, no. 1, pp. 42-52 (2004)

NNGen: a powerful tool for the implementation of Artificial Neural Networks on a chip

Tosini, Marcelo[1] Acosta, Gerardo[2]

[1] INTIA/INCA - Grupo de Investigación en Computación Aplicada
Fac. de Ciencias Exactas - Universidad Nacional del Centro de la Prov. de Bs
Paraje Arroyo Seco s/n
Tandil - Argentina
mtosini@exa.unicen.edu.ar

[2] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) - Grupo INTELYMEC (ex-ADQDAT)
Dept. de Ing. Electromecánica – Fac. de Ingeniería - Universidad Nacional del Centro de la Prov. de Bs. As. (UNCPBA)
Av. del Valle 5737 – (B7400JWI) – Olavarría- Argentina
ggacosta@fio.unicen.edu.ar – URL: http://www.fio.unicen.edu.ar/Curriculum/Acosta

Abstract

A design and development tool to achieve artificial neural networks (ANN) implemented in a Field Programmable Gate Array (FPGA), is presented in this article. Its main components and functionality are thoroughly described. This tool, called NNGen, allows constructing digital ANN, which are easily programmable selecting different parameters. The output of such programming task is directly VHDL code to be ported to, in principle, any chip of the mentioned kind. A case study of a multilayer perceptron applied to weather forecast that was fully designed with NNGen, is also analyzed to obtain some conclusions.

ANN on a chip: an introduction

Artificial neural networks (ANN) can be classified into two groups based on the degree of parallelism obtained in the diverse developments. Developments with high degree of parallelism implement the architecture of the network with all their neuronal interconnections [6]. These developments are limited to small networks consisting generally of one hidden layer with less than 10 neurons.

On the other hand, developments of low (or null) level of parallelism completely implement the functionality of a single neuron plus a control unit. That unit successively feeds the neuron with sets of weights and input values to obtain a valid output result. This last alternative allows implementing time multiplexed hardware circuits with a high advantage in area without loosing much performance.

Time Multiplexed neural networks are commercially available whereas other designs, like replication of various neurons, are being studied by many research and academic groups.

The two main categories consist of neurocomputers based on standard integrated circuits and ASIC. The first ones are accelerator boards that optimize the speed of calculation in conventional computers (PC like or workstation). In these cases, where standard components are used, the designers can be concentrated totally in the development of a particular technology. In the second ones, several alternatives and technologies of implementation can be chosen for the neuronal accomplishment of chips, like digital, analog or hybrid neurochips.

Direct implementation in circuits generally alters the operation of the original processing elements (analyzed or simulated). It is due to the limitation in precision. The influence of this limited precision is of great importance for the correct operation of the original paradigm. Because of this, many designers have dedicated much time to study these topics. In order to obtain implementations on great scale, several neurochips must be interconnected to create systems of greater complexity, with advanced communication protocols.

Proposed architecture for digital ANN

In the present approach an architecture of a Digital ANN is proposed. It has four main functional components: Data memory, weights memory, neuron module and control unit. All these components have a generic orthogonal structure in order to facilitate automatic generation from specific parameters. Figure 1 shows ANN building block with the interconnection buses and control signals [5].

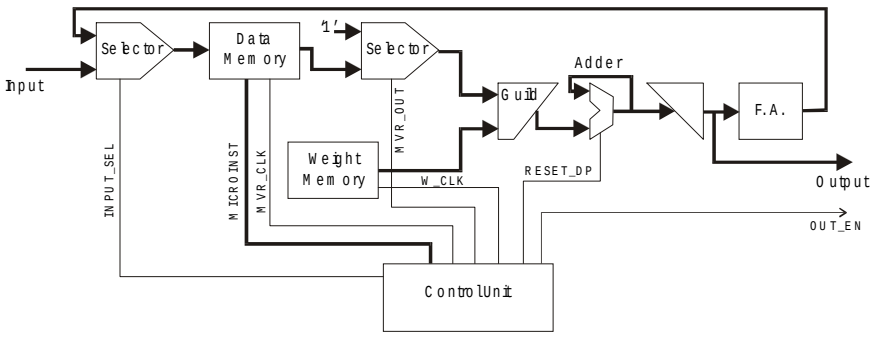


Fig. 1. Developed ANN building blocks

In a particular ANN design all of its components are generated with specific size, depending on two types of parameters: topological and architectonic. The first ones (i.e. number of inputs and number of hidden neurons) will determine the size of the memories, the neuron processing time and the microprogram size. The second ones (i.e. word size in bits and circuit granularity) will determine the precision of the results and the final ANN performance. This parameter definition has influence in the final circuit size, so the designer must take care of this because the programmable devices (FPGA, CPLD) – the physical support of the ANN, is

space limited. A commonly used metric hint compare areas of different ANN is the number of physical registers in the circuit. Equation 1 shows this metric for an ANN with *n* inputs, *m* hidden neurons, word size of *w* bits and circuit segmentation granularity β.

$$Nr_{total} = \frac{2}{\beta}.w^2 + \left(7 + 2.\left\lfloor \frac{2w-2}{\beta} \right\rfloor\right).w + \left\lfloor \frac{2w-2}{\beta} \right\rfloor + 6 + (n+m-1).w + \quad (1)$$

$$+\left\lceil \log_2 (m.(n+2)+1) \right\rceil$$

The proposed architecture is programmed according two internal sub-parameters calculated from the parameters described previously. Once this step is fulfilled, a microprogram should be written to control the computations of the particular design. This microprogram consist in a variable number of micoinstructions depending on ANN topology. Each microinstruction configures all datapath signals for each clock cycle, whereas the entire microprogram, cyclically executed, allows obtaining subsequent decisions of the network.

Parameter Definition

The first parameter to consider is the number of cells of the Data Memory (***L***), according to:

$$\boldsymbol{L} = \max (\boldsymbol{N} - 1) \quad (2)$$

where ***N*** is the number of neurons in two contiguous level. This Data Memory must always be able to store the outputs of the just computed level (data) plus the results of computing each neuron in the present level. The second parameter is the size of the weight's memory (***S**wm*), which shall store all of the ANN's weights.

Example I: a classical feedforward ANN of nine neurons in the hidden layers and four inputs and three outputs is described according Fig. 2.

Microprogram creation:

From two levels of abstraction, a software tool (assembler) was developed to analyze the precedence relationships in the computations needed by the network under design. The higher level of abstraction generically describes which kind of neural network is under development. The present types supported by NNGen are Multilayer Perceptrons (MLP) and Hopfield ANN, synchronic and asynchronic [7].

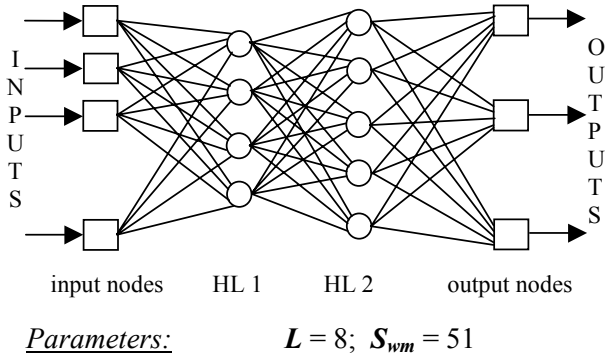


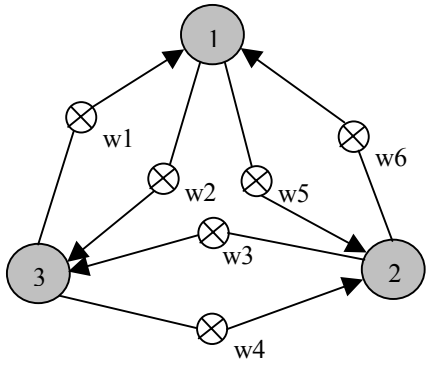
Fig. 2. Parameters definition of an ANN

The feedforward ANN of Fig. 2 is defined as:

```
Net <name> Description
      type feedforward
      Input : 4
      Output : 3
      Hidden : 4 , 5
End <name>
```

Example II: a given Hopfield ANN is defined as:

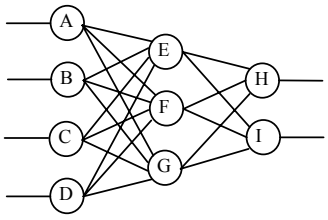
```
Net <name> Description
  type sync Hopfield
  Nodes : 3
End <name>
```



The synchronic or asynchronic feature of the Hopfield ANN is achieved by the microprogram taking into account that the neurons are all updated in each clock cycle or sequentially, respectively. This high level description only allowed the generation of simple and remarkably regular neural nets, but did not permit to outline more specific details about the neurons interaction of a particular design. Then a lower level of description was needed, in which neurons dependencies were explicitly stated. This was achieved by programming in the form shown in Example III.

Example III: a more detailed description of a feedforward ANN with 4 inputs, 2 outputs and a hidden layer of 3 neurons, in a lower level of abstraction

```
Net <name> Description
  type Feedforward
    Nodes: A,B,C,D,E,F,G,H,I
    Weights: w1..w18
    Input Nodes: A,B,C,D
    Output Nodes : H,I
  Relation
    E = A*w1 + B*w2 + C*w3 + D*w4
    F = A*w5 + B*w6 + C*w7 + D*w8
    G = A*w9 + B*w10 + C*w11 + D*w12
    H = E*w13 + F*w14 + G*w15
    I =  E*w16 + F*w17 + G*w18
  End Relation
End <name>
```



From this description, the assembler can settle down the introduced parameters $\boldsymbol{L}$ (equation (8)) and $\boldsymbol{S_{wm}}$ that for this example will be 6 and 18 cells, respectively. The microprogram generation is based on these specifications, which maps into a set of microinstructions that the control unit successively stores in the

corresponding register. This register has 2***L*** bits, coding one among the four possible operations for each cell of the data/results row. They are presented in table I.

The resulting microprogram of Example VI is partially shown in table II. The microprogram puts in the data row the resulting values of computing each neuron, in a position such that this value operates with the corresponding weight when going out of the row. To achieve this, weights are stored in a circular buffer of ROM memory. In this way, the ANN behaves like a systolic system, multiplying in each clock cycle a value from the data/results row and a corresponding synaptic weight. This product (a partial result) is stored until every input to each neuron is processed. Once the output of a processed neuron is obtained, it is crossed through the activation function and the result is stored in the data row, in the cell pointed by the current microinstruction. This procedure is illustrated in Fig. 3.

Table 1. microinstructions of the microprogram

Rotation (C)	*The cell receives the row output value*
No operation (N)	*The cell is not modified*
Shift (S)	*The cell receives the value from its prior one (left)*
Datum load (L)	*The cell is loaded with the value from the activation function or the input data*

Table 2. part of the microprogram of Example VI

Cycle	μicroinst.	Row state	Operation	Comments
0	NNNNNL	_ _ _ _ _ A	_	*initial data load*
1	NNNNLN	_ _ _ _ B A	_	
2	NNNLNN	_ _ _ C B A	_	
3	NNLNNN	_ _ D C B A	_	
4	NNCSSS	_ _ A D C B	Acc = A * w1	*A is recycled*
5	NNCSSS	_ _ B A D C	Acc += B * w2	*B is recycled*
6	NNCSSS	_ _ C B A D	Acc += C * w3	*C is recycled*
7	NLCSSS	_ E D C B A	E = Acc += D * w4	*D is recycled and E is loaded*
	.....	.....	.....	
11	SLSSSS	_ F E D C B	F = Acc += A * w4	*shift and F is loaded*
12	SSSSSS	_ _ F E D C	computation of G	*global shift*
	.....	.....	.....	

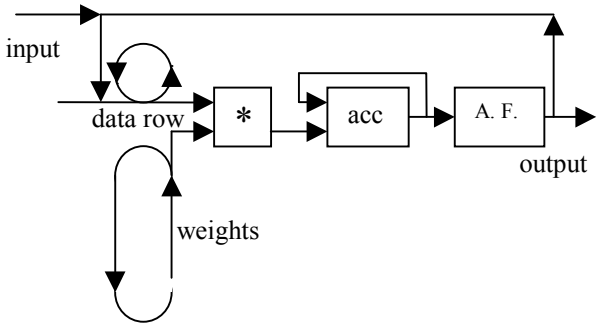


Fig. 3. Data flow in the ANN

Description of NNGen

A complete set of tools -NNGen- was developed to assist the user in the ANN generation and test. NNGen helps the user in the creation and handling of different projects in order to maintain diverse designs of MLPs and Hopfield ANNs [8]. For each project, it is possible to specify the topographic and physical parameters to personalize the circuit, generate the control program, simulate the microprogram operation and, finally, generate VHDL source code for all the components. Figure 4 shows the main window with all the necessary functionality to develop a project.

MLP topographic parameters are three: number of inputs, hidden layer dimension (number of layers and neurons per layer) and number of outputs. These parameters allow the system to generate the control microprogram to drive all datapath signals in each clock cycle.

Physical parameters allow the system to adjust all datapath-operators size and to generate data and weight memories (RAM and ROM respectively). The parameters are: a) Weights file, necessary for 'off-line' training; b) Word-size, to determinate internal data size and c) Working precision for current application, in order to establish the Arithmetic to use. For example, the arithmetic showed in figure 5, *(4,4)* set the datapath to work in 8-bit with *eeee.dddd* format.

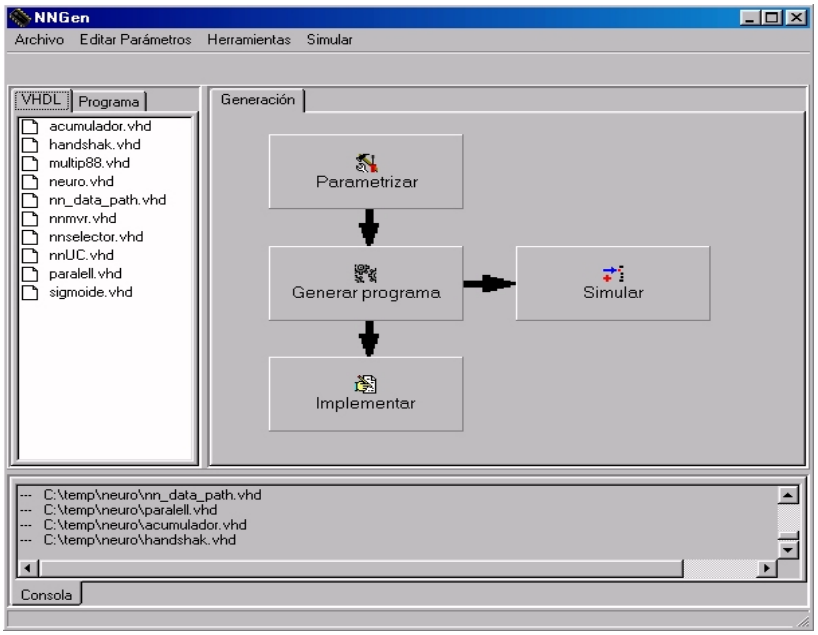


Fig. 4. NNGen tool main window

For a generated microprogram, simulation tool allows the user to make logic simulations to test the correct datapath operation. The signals synchronization to handle the systolic datapath is highly complex. For this reason, the simulator allows following the evolution of the calculations in all the points during each clock cycle. Figure 6 shows the microprogram simulation-tool used to test a MLP with 4 inputs and 2 hidden neurons.

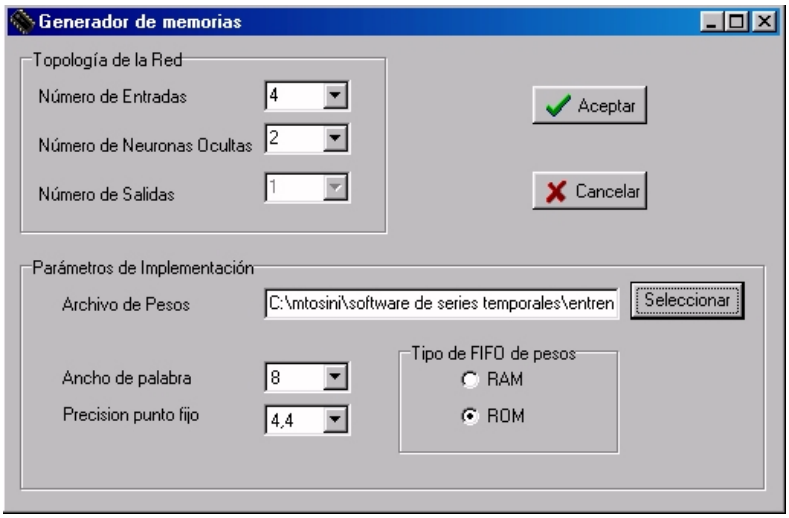


Fig. 5. Parameter required for ANN generation

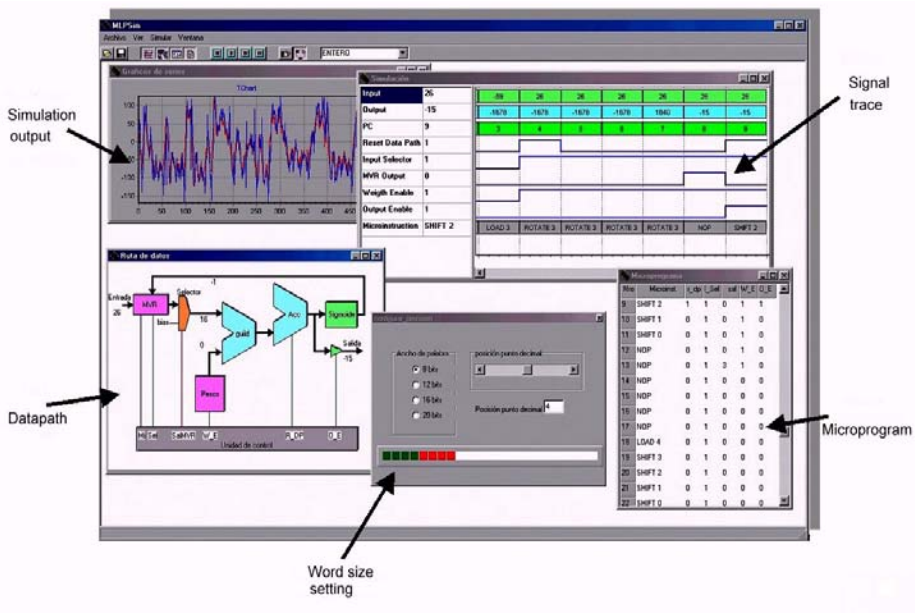


Fig. 6. Logic simulator to test the microprogram funcionality

Temperature forecast: a case study

One of the main problems to solve in temperature forecasting is to achieve the ability of prediction of time series. The ANN approach seems attractive in this task from several points of view [1], [2]. Effectively, there are various ANN architectures, capable to learn the evolutive features of temporal series, and by so doing, to predict future states of these series from past and present information. Perhaps the most used ANN architecture for this kind of prediction is the MLP, like the one in Fig. 7 [3].

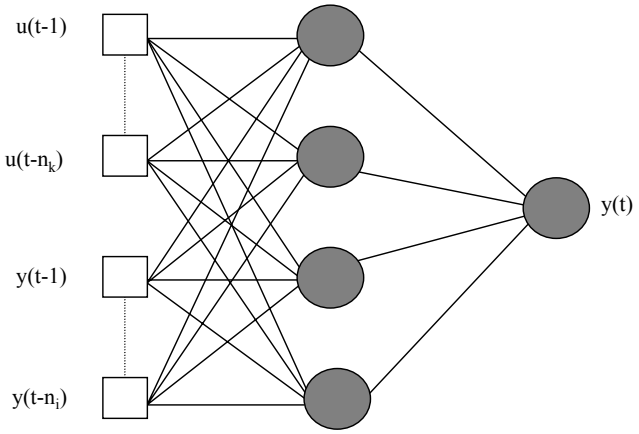


Fig. 7. The multilayer perceptron (MLP)

The input vector for such MLP will consist of a certain amount ***p*** of past samples of the temporal series, of the form:

$$x = [x(n-1), x(n-2),.., x(n-p)] \quad (3)$$

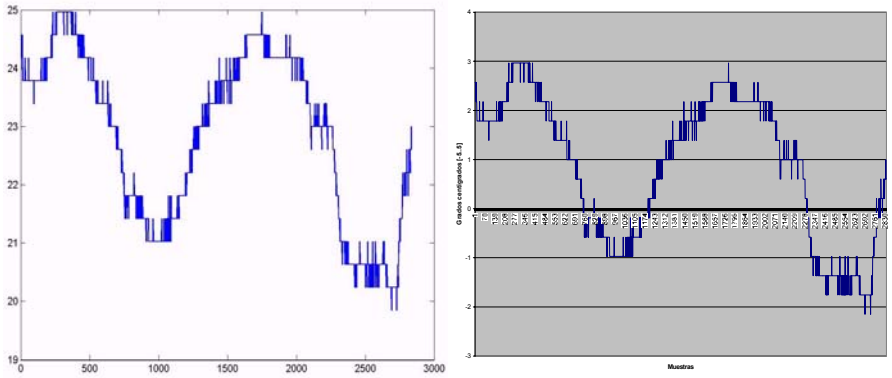
The net output will be a variable [1]:

$$y(n) = \varphi[x(n)] \quad (4)$$

If φ is static, then (3) and (4) are only applicable to stationary temporal series. This is a drawback for weather forecasting. Then, a reasonable choice is to use some kind of memory to store relevant temporal information, like in Elman neural networks or FIR (Finite Impulse Response) neural networks. The first uses a positive feedback through memory units called "context memory". This memory allows adding past behaviors of the net to the learning process. On the other hand, in FIR neural networks, FIR filters replace the primary synapses weights, also with the added feature of temporal recalling. This change yields a vectorial and temporal extension to the MLP. Of course depending on the application, if the prediction horizon is small enough, temporal series may be considered as stationary. However, measuring in time units this prediction horizon for weather forecast applications, is still a matter of research and experimental studies.

Putting NNGen to work

A temporal series was used to train four topologies of MLP as a testbed. Figure 8-a) shows original series consistent of 2838 samples of environment temperature in a lapse of 47 hours 18 minutes in intervals of 1 minute. Figure 8-b), shows the same series normalized into range [-5º..5º] (SERIE I).



(a) (b)

Fig. 8. (a) Original temporal series with temperatures from 13:34:40-15/12/2001 to 12:51:40-17/12/2001. (b) Normalized temporal series in range [-5º..5º]

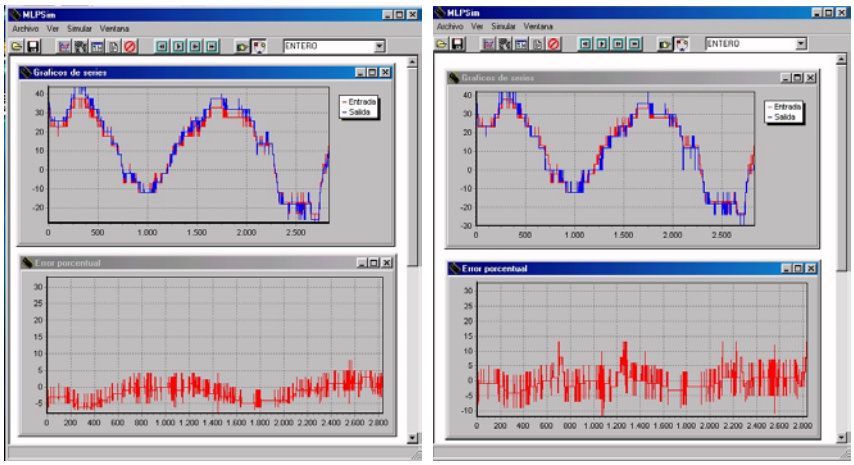
The implemented MLPs consisted of 4 inputs and 5, 4, 3 y 2 neurons in the hidden layer respectively. They were trained [4] using the first 1500 samples of SERIES I.

Hardware simulation for the four cases are showed in table III, where the minimum, maximum and average errors are appraised (generated minus expected value).

Table 3. Hardware simulation of cases

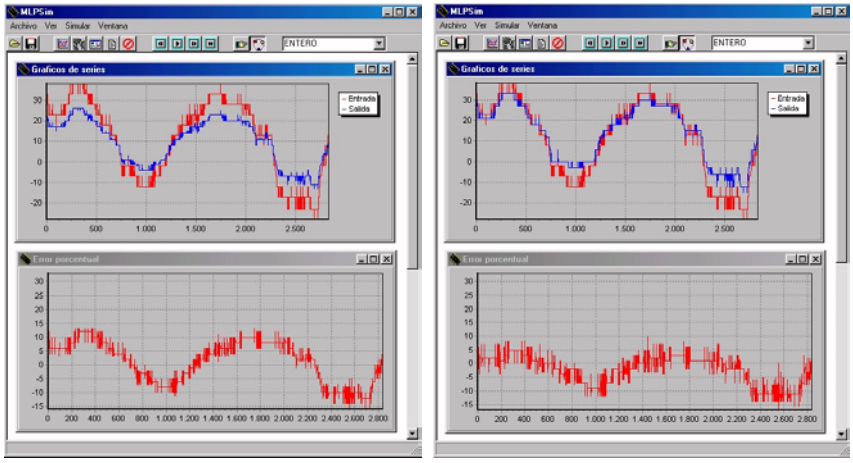
Case	Topology Inputs/hidden	Probe stage	Error		
			Maximun	Minimun	Average
I	4 / 5	SERIE I 2838 samples	11 0,85º	0 0º	1,54 0,12º
II	4 / 4	SERIE I 2838 samples	8 0,625º	0 0º	2,47 0,19º
III	4 / 3	SERIE I 2838 samples	17 1,32º	0 0º	4,26 0,33º
IV	4 / 2	SERIE I 2838 samples	16 1,25º	0 0º	6,59 0,51º

Case I (5 hidden neurons) is the best solution, with an average error of 0,12º. **Case II** (4 hidden neurons) is also a good solution as regards as error, with the added advantage of a lower cost in area than the previous case. Figure 9 shows hardware simulation for cases I and II, while figure 10 shows cases III and IV, all of them using the hardware simulation tool of NNGen.



a) Case I b) Case II

Fig. 9. Simulation for cases I and II



c) Case III d) Case IV

Fig. 10. Simulation for cases.III and IV

Conclusions

Hardware architecture for digital ANN design was developed with the main features of an easy configuration from a small set of parameters, a pipelined datapath and a microprogramed control unit. These characteristics have two great advantages: allowing resources reusability and achieving high speeds of operation.

In order to construct the ANN a set of software tools (NNGen) was developed to offers to the user a simple and friendly interface for design and test purposes. Among the outstanding features of such tool are generic VHDL generation, hardware simulation capabilities, and dynamic re-configuration.

In this article, this tool was employed to develop different topologies of MLP which were trained and probed with several temporal series of temperatures with depreciable prediction errors.

Future works includes development of algorithms for automatic generation of other paradigms such as Hopfield networks, Radial Basis Function networks and K-Nearest Neighbor (KNN); and on line MLP training.

References

1. Koskela, T. Lehtokangas, J. Saarinen and K. Kaski, "Time Series Prediction With Multilayer Perceptron, FIR and Elman Neural Networks", Proceedings of the World Congress on Neural Networks, INNS Press, San Diego, USA, pp. 491-496, 1996.
2. Corchado J., Fyfe C. y Lees B., "Unsupervised Neural Method for Temperature Forecasting", Proc. of the Int'l ICSC Symposium on Engineering of Intelligent Systems EIS´98, Volume 2, Neural Networks, pp. 300-306, 1998.
3. Haykin, S., "Neural Networks – a Comprehensive Foundation", Prentice Hall, 2nd Ed, 1999.
4. Hagan, Martin T. y Menhaj, Mohamed B., (1994), "Training Feedforward Networks with the Marquardt Algorithm", IEEE Transactions on Neural Networks, Vol. 5, No. 6.
5. Tosini, Marcelo, "Hardware Implementation of a segmented data route for Neural Networks" (in Spanish), VI Workshop Iberchip, IWS´2000, Sao Paulo, Brazil, pp. 292-299, 2000.
6. Nordström, T. and Svensson, B., "Using and Designing Massively Parallel Computers for Artificial Neural Networks", Journal of Parallel and Distributed Processing, vol. 14, no. 3, pp. 260-285, 1992.
7. Gurney, K., "Computers and Symbols Versus Nets and Neurons", UCL Press Limited, UK, 1995.
8. Tosini, Marcelo, "Sistema basado en Redes Neuronales Digitales aplicado a la predicción climática en Ambientes con microclima controlado", Tesis work for the degree of Magister in system engineering, 2002.