Fast Prototyping of Artificial Neural Network: GSN Digital Implementation

Eduardo do Valle Simões, Luís Felipe Uebel & Dante Augusto Barone
Curso de Pós-Graduação em Ciência da Computação - Universidade Federal do Rio Grande do Sul
Porto Alegre, RS - Brazil Caixa Postal 15064 - CEP 91501-970
E-mail: EDSIM, UEBEL, BARONE@inf.ufrgs.br

Abstract

This work describes a framework for a GSN (Goal Seeking Neuron) Boolean neural network fast prototyping into an user-programmable gate array. This system provides a VHDL language description of the trained network, allowing the direct implementation of the circuit on an academic FPGA (Field-Programmable Gate Array). A GSN software tool was designed to train and simulate an user-defined network, with diverse dimensions and applications. The implemented network presents 60 neurons in four pyramids with four layers. The short propagation time (30 ns) of the network output provides the requirements to deal with real time neural applications

1. Introduction

Artificial Neural Networks (ANN) have made possible the solution of disparate problems more easily than the solutions previously presented to problems such as: pattern recognition, image processing and so on. All these applications need a large amount of processing capacity, which is made possible only with the development of specific architecture. The design of suitable neural architecture is imperative to reach good performances at reasonable economical costs.

The development of ANN in analog [1] or digital [2] integrated circuits, is in general bounded by the need to generalize the functions performed by the designed circuit, aiming its utilization in a wider range of possible applications. This generalization contributes to economical factors, nevertheless it diminishes the natural network size and its performance, limiting its application in real time processing substantialy.

The utilization of a programmable logic circuit to implement a neural network becomes very attractive since it allows fast hardware designs. It also permits to make design modifications at low costs. Among the programmable logic circuits, the Field-Programmable Gate Arrays (FPGA) represent the most attractive option to implement neural networks.

This work presents a fast prototyping system for Boolean neural networks. That system enables the designer to define the structure of the neural network and the pattern to be learned. It also performs the simulation of the ANN, helping to choose a particular architecture. It delivers a VHDL description, which is taken as input to a FPGA prototyping tool. In a previous work, the design of a Boolean GSN ANN full custom circuit, named DIANNE [3-4] was shown. The development of this circuit has shown the authors the importance of developing a flexible hardware to implement different ANN configurations without disrupting performance. In this sense, considering just the GSN Boolean model, we tried to use an academic FPGA matrix, named FLECHA, to prototype an ANN based on this model.

The article contains a brief description of the GSN model, a description on how the proposed ANN system works, a description of the employed methodology in the implementation of the neural networks, a brief presentation of FLECHA programmable logic cell matrix, and finally, the results obtained from the prototyping of the GSN neural network circuit in the FLECHA matrix.

0-8186-6710-9/94 $4.00

2. Presentation of the FLECHA Matrix

The FLECHA matrix and all of its characteristics try to provide control logic between microprocessors and memories, where gate complexity is not too high, but the number of I/O signals that need to be processed is. This application leads to the ideal topology of the FLECHA matrix the one that presents the same number of logic cells and I/O cells. This is an efficient way to get the balance between the implementation capacity and the great demand of big interconnections required by glue logic applications.

The general topology of the FLECHA matrix is shown in figure 1, and it has 40 programmable logic cells distributed in two columns of four rows or groups with 5 logic cells and 5 I/O pads that can communicate through the central bus to implement complex functions. This figure presents the regularity of the chip layout, developed with ES2 CMOS 1.2 μm technology. This topology allows the array to implement a function with up to five cells, or many small functions in each row. There are cases in which one needs to use more than five cells to implement a complex function, then one can connect two or more groups of 5 cells in the matrix by the central bus. Thus, user-programmable logic cell arrays are viable alternatives to conventional mask-programmed gate arrays in most applications. However, since the FLECHA matrix is a standard user-programmable product, it does not suffer from the costs and risks of mask-programmed devices; there are no NRE (Non-Recurrent Engineering) charges, no test program development, no inventory risk, and no schedule risk due to design changes [9-10].

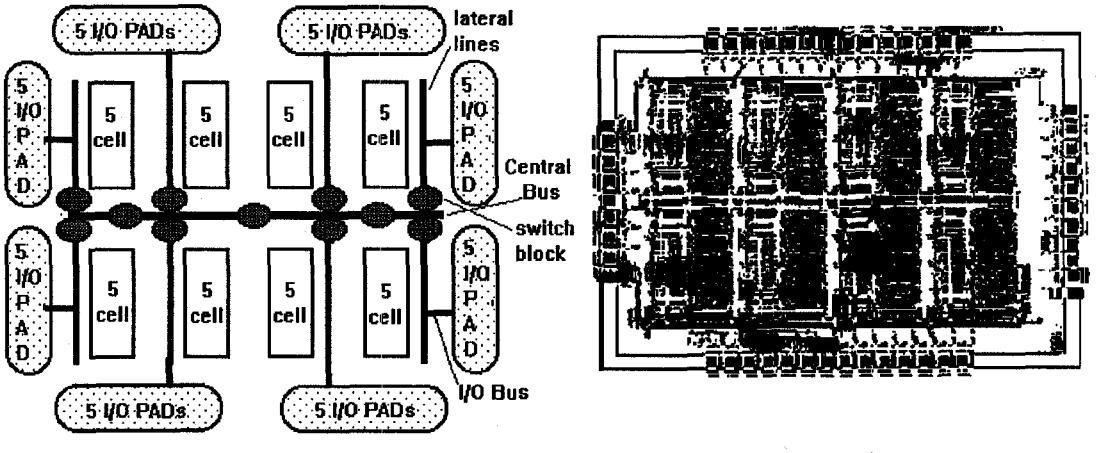

Fig. 1: Layout and general topology of the FLECHA matrix.

2.1. Design Considerations

The architecture of FLECHA matrix has three distinct elements: the programmable logic cells, the configurable I/O pads, and the interconnection network. Five logic cells are placed in groups with five I/O pads, and these groups are regularly distributed by the matrix. This solution provides a direct communication between pads and the internal logic cells.

Figure 2 shows the architecture of a three-input logic cell that provides the functional elements, with which the logic of the user is constructed. In this logic cell, a shift-register chain and a 8:1 multiplexer are used to perform the truth table of the functional block of the cell. The logic cells of FLECHA are able to implement any Boolean function of up to three inputs, representing the function truth table in the internal static memory cells.

A 2:1 multiplexer can connect an internal register to the output of the functional block if it is necessary to implement sequential logic, as it is shown in figure 2. This register is an edge-triggered D-type flip-flop. Thus, the 40 logic cells are capable of implementing any 40 independent sequential functions of up to 3 input variables. Four multiplexers are used to connect the three inputs and the output of the cell to the lines of the data bus, which is used to interconnect each cell side by side as shown in figure 2.

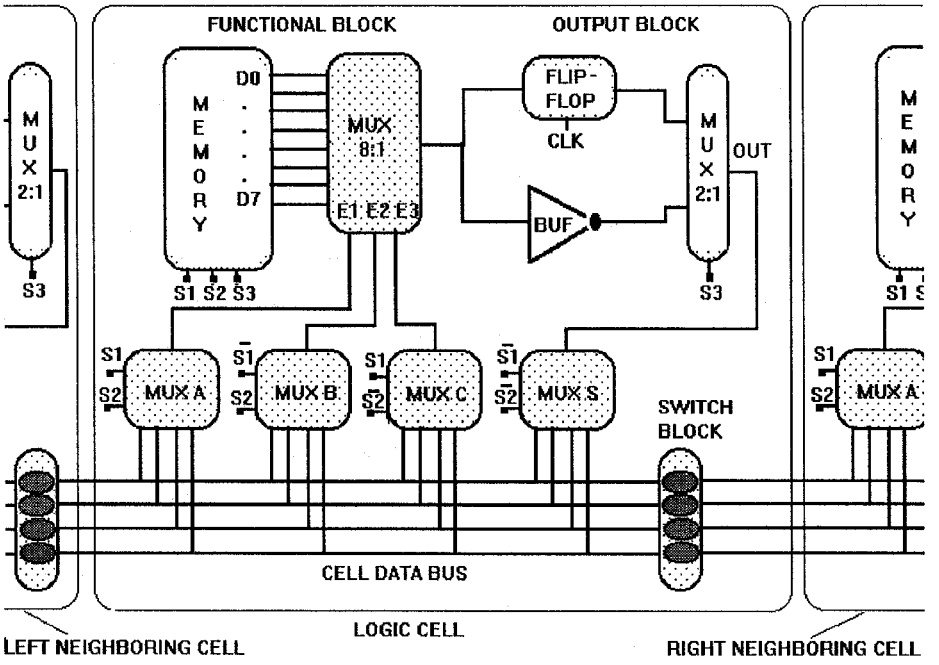

Fig. 2: The architecture of a three-input logic cell.

The function of the logic cells, the configuration of the I/O pads, and the routing of the interconnection network are defined by a configuration program stored in the internal static memory cells. These cells consist of a shift-register chain that can be loaded automatically at power-up. The process of loading the configuration memory is independent of the logic functions of the user that are implemented in the matrix.

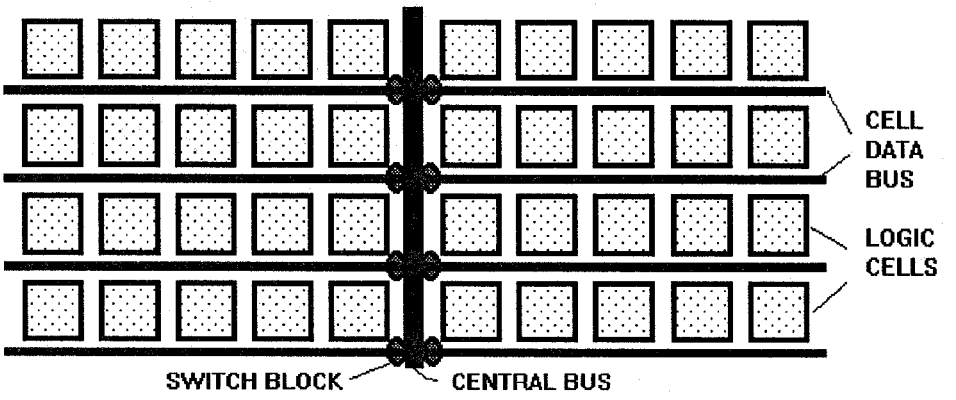

Fig. 3: The row placement technique applied to a 40 cell matrix.

Bi-directional pads are connected to their control circuits to provide the configurable input/output blocks. The I/O pad architecture allows the separation between the pad proper circuit (buffer and filter) and the configuration structure. Thus, the pad circuit can be re-designed separately, in case a new process technology becomes available.

2.1.1. Internal Interconnection Network

The interconnection topology of FLECHA was designed according to the row placement principle, developed to reduce the amount of switches without reducing the interconnection capacity of the logic cells. This technique consists of placing all the cells that implement the same logic function side by side (it means

that a powerful software tool will be necessary to place and route these cells). Once they are placed in rows (see figure 3), those cells need only to exchange signals with their neighbors, and this fact permits a great simplification in the switching structures. If it is necessary to use more than one row of cells to implement one big logic function, a new row, under or above, can be connected by the central bus utilization. The details of the innovative placement strategy are emphasized in reference [10].

2.2. Performance Considerations

The 48 pin DIL-type package was chosen because of its good relation between costs and available pins. This package allows the design of a matrix with 40 logic cells and 40 I/O pads, performing 600 equivalent gates. Any Boolean function can be implemented in each logic cell, where a large number of registers are provided, and any necessary routing paths can be defined between the logic cells and the I/O pads.

Programmable gate arrays are assigned a *speed grade* based on the maximum toggle rate of the internal flip-flop of a single logic cell [9]. The matrix operational frequency, due to its simplified internal structures, is about 145 MHz, as a single logic cell delay is less than 6.0 ns. The external frequency is often smaller due to the delay of the I/O pads. With the pads of FLECHA matrix, this external frequency is 66 MHz.

3. Tools for ANN CAD

The design environment is divided into five different phases: ***i***) sizing of the neural net by the user; ***ii***) training of the net with the patterns delivered by the user; ***iii***) logic mapping of each neuron of the net, which is treated as a *black-box*; ***iv***) generation of the VHDL description of the complete GSN neural net, with all pyramids and neuron circuits already trained to the FPGA design system; ***v***) simulation of the behavior of the neural net. These five steps will be described after a presentation of the GSN model.

3.1. Brief Description of the GSN Model

The GSN model was proposed by Edson Filho *et al.* [5-7]. It is derived from the PLN model of Kan and Aleksander [8]. The GSN differs from the PLN model because it can present in its output an indefinite state **U**. The network consists of pyramids that can be connected in different organizations, making IC implementations easier.

This type of network has many advantages over other neural network implementations: ***i***) it presents a lower number of interconnections between neurons; ***ii***) its simple structure permits asynchronous implementation of neurons, allowing massive parallel processing; and ***iii***) its pyramidal regular structure gives good flexibility to the designer.

The GSN neural network has three different states of operation: the *Validation*, the *Learning* and the *Recall Modes*.

The goal in the *Validation Mode* is to stabilize the network in an indefinite state. In this mode the neuron validates the possibility of learning without disrupting stored values, selecting appropriate memory positions for this possibility.

The output of this mode is given by: ***1*** iff all the selected memories are ***1***; ***0*** iff all the selected memories are ***0***; and ***U*** for all other values of the selected memories. If the pyramids compute the indefinite value ***U***, they are not saturated for this pattern and have capacity for learning.

The *Learning Mode* aims to store appropriate values in memory positions selected in the *Validation Mode*. These values are stored only in memory positions having an indefinite value ***U***.

The goal in the *Recall mode* is to stabilize on the values ***1*** or ***0***. The indefinite value **U** will not be propagated and the patterns will be recognized among learned patterns.

Figure 4 presents the learning and validation modes of the network. The memory positions with the signal (+) will change their context to acquire the pattern being learned. The signals (*) show the positions of the selected memories in the validation mode.

The output of this mode is given by: ***0*** iff the number

of *0* values in selected memories is greater than *1*; *1* iff the number of *1* values in selected memories is greater than *0*; and *U* iff the number of *1* and *0* is equal.

After the one-shot learning provided by the GSN supervised learning algorithm, consisted of the validation and learning modes, the network is ready to classify patterns previously learned. The network becomes ready to work in the recall mode.

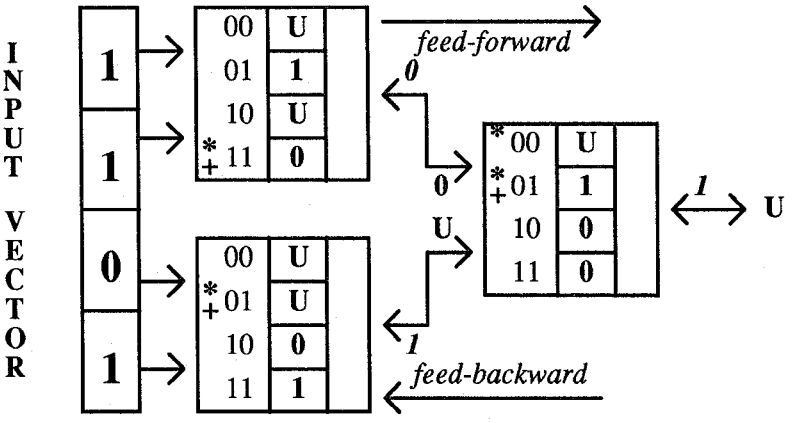

Fig. 4: Validation and Learning Modes in GSN Neural Network.

3.2. Sizing of the Neural Net

The sizing of the neural net is made by the user, who specifies the software the configuration of the neural net.

The definitions made by the ANN designer include: *i*) number of inputs of the net, that is, the number of inputs each pyramid may have is specified. That number does not need to be equal to the number of inputs of the considered pattern; *ii*) number of levels of each pyramid; *iii*) number of inputs of each neuron. The specification of the number of levels and inputs of the neurons produces a greater uniformity to the saturation for the learning of new patterns. Once the user has made all those described definitions, the system automatically builds the neural net.

3.3. Training of the Net

In this phase, the system teaches the neural network to recognize the patterns delivered by the user following the rules presented before:

The user himself can define the output pattern to each input pattern previously presented to the network through a file. The training of the net includes the validation and learning modes, which permit to identify the memory locations that can learn a new pattern presented to the net, and effectively teach the net how to acquire a new pattern.

The user has some flexibility, such as: *i*) to allocate just one pyramid to a pattern, which allows a wider utilization of the net; *ii*) to code the output pattern in binary notation to all pyramids, which allows up to 2^n different patterns to be learned, where n is the number of pyramids in the net.

3.4. Logic Mapping of the Neurons

Each neuron of the net is mapped as a *black-box*. The system excites a neuron with all possible input combinations and identifies the corresponding output to each excitation. With this procedure, it is possible to construct a truth table that leads to a logic description of the neuron. This logic description gives rise to a schematic diagram of each neuron. This operation is extended to all neurons of the network. Each input/output of a neuron must be represented by two bits, due to the *U* value. We have then the *0* logic value, represented by 00, the *1* value, by 10, and the *U* value, by 01.

The neuron modeled by a *black-box* is shown in figure 5. This procedure simplifies the logic synthesis of the neuron and diminishes the power consumption of the logic gates. The programming of the neural network becomes implicit to the logic mapping, allowing no waste of logic connections, enabling a better performance of the

system.

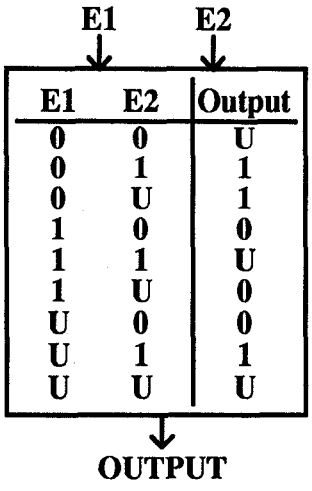

Fig. 5: *Black-box* representation of the neuron.

3.5. VHDL Description

Once the logic mapping of each neuron is completed, the system delivers the user a VHDL description of the mapping of the whole network. This enables the prototyping of the net. The VHDL description permits a direct interaction with the FPGA prototyping systems, which are responsible for the logic synthesis of the circuit.

The VHDL description includes non-minimized logic mapping of all neurons of the neural network, following the sizing of the net, previously defined by the user.

3.6. Network Simulation

The system allows the recall mode simulation with noise inclusion (the inversion of some bits) in the user proper patterns or the ones used to train the neural network. The neural network simulation provides the detection and improvement of the learning of the system, aiming to increase the network recall level.

The recall level depends on the similarity degree of the patterns taught to the system and the number of pyramids. The recognition capability of the system increases with lower similarity degrees between patterns, and greater number of pyramids. With the utilization of the simulator, a preliminary evaluation of the recall level is made possible, permitting a fast test and validation of possible design modifications.

Figure 6 shows the flow chart of the neural network design environment. The simulation module permits modifications of the entire set of input patterns to calculate the size of the network by the user, in an interactive form.

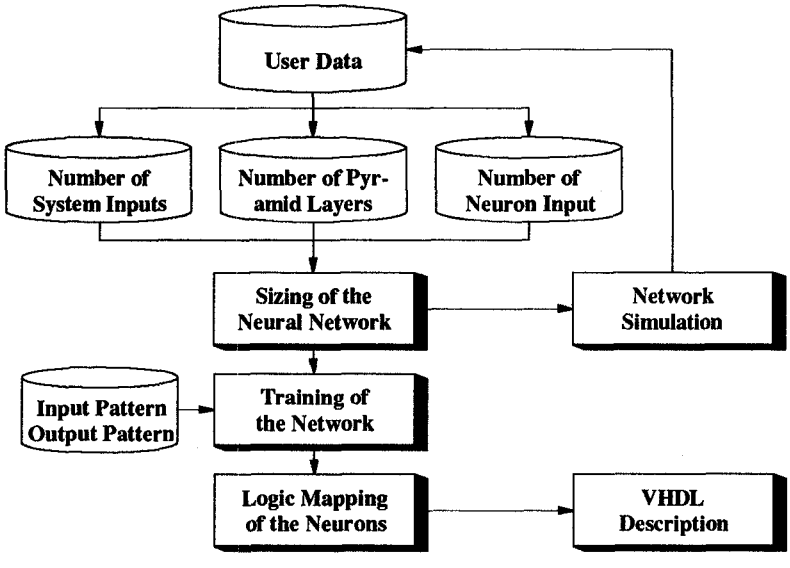

Fig. 6: The flow chart of the framework for neural networks design.

4. Automatic Integrated Circuit Generation

A GSN neural network was implemented by the first time in the DIANNE circuit [3-4]. This circuit, having 60 neurons grouped in 4 pyramids, has 17 thousand transistors. The DIANNE circuit stores the weights of the neurons in the internal memory cells and processes its outputs according to these weights and the pyramid inputs.

Table 1: Truth table of the neurons showed in the pyramid in figure 7.

		INPUTS								
LAYER	**NEURON**	**00**	**01**	**0U**	**10**	**11**	**1U**	**U0**	**U1**	**UU**
1	A0	0	0	–	U	0	–	–	–	–
1	A1	0	U	–	U	0	–	–	–	–
1	A2	U	1	–	U	0	–	–	–	–
1	A3	1	U	–	1	1	–	–	–	–
1	A4	0	U	–	1	U	–	–	–	–
1	A5	0	U	–	U	U	–	–	–	–
1	A6	1	U	–	U	1	–	–	–	–
1	A7	0	1	–	1	U	–	–	–	–
2	B0	0	U	0	U	U	U	0	U	0
2	B1	U	0	0	U	1	1	U	U	U
2	B2	1	U	1	0	U	0	U	U	U
2	B3	U	U	U	0	1	U	0	1	U
3	C0	1	0	U	U	U	U	1	0	U
3	C1	0	0	0	1	0	U	U	0	0
4	D0	U	1	1	0	U	0	0	1	U

The same topology of the DIANNE circuit was implemented in the FLECHA matrix too. In that case, the development took a different way. With the utilization of the presented system, a VHDL description of each neuron was provided. The weights of the neurons were not stored in the internal memory cells, but they were directly mapped in the VHDL description of the neuron. This solution provides a minimization of the logic gates, since only the significant connections are taken into account. It reduces the circuit flexibility, since it has to be redesigned if the network has to learn a new pattern.

Figure 7 presents one of the four pyramids of the neural network. The neurons of the first layer are marked with letter "**A**", the neurons of the second, with letter "**B**" the ones of the third, with letter "**C**", and the neuron in the fourth layer is marked with letter "**D**". The inputs that are accepted by the network are only the ***0*** and ***1*** values. It provides the reduction of the input bus dimensions. Table 1 shows the *black-box* treatment of the same pyramid. Note that the first layer neurons do not have ***U*** inputs.

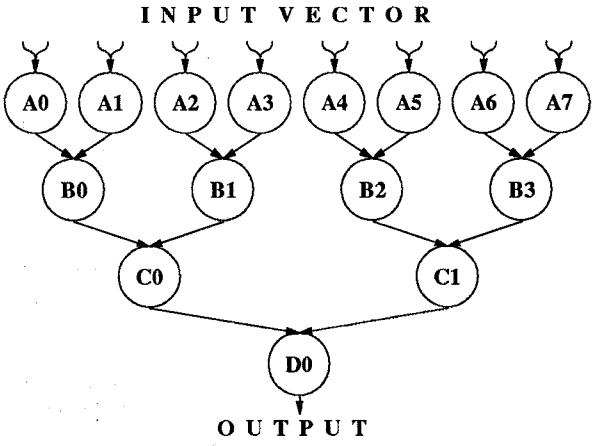

Fig. 7: The organization of a GSN neural network pyramid.

Table 2 presents the VHDL description of the neurons shown in figure 7. Note that we need two bits to represent the three possible states of each neuron. The inputs of that pyramid are represented by letter "**e**". The "**!**" symbol is equivalent to the logic ***NOT***, "**&**" to the logic ***AND***,

and the symbol "#" to the logic ***OR***. The processing time of the neural network design tool was about 1 minute.

Table 2: VHDL description of the neurons presented in figure 7.

LAYER	NEURON	VHDL DESCRIPTION OF EACH NEURON
1	A0	A00 = GND; A01 = e00 & !e01;
1	A1	A10 = GND; A11 = !e10 & e11 # e10 & !e11;
1	A2	A20 = !e20 & e21; A21 = !e20 & !e21 # e20 & !e21;

1	A3	A30 = !e30 & !e31 # e30 & !e31 # e30 & e31;
		A31 = !e30 & e31;

1	A4	A40 = e40 & !e41; A41 = !e40 & e41 # e40 & e41;
1	A5	A50 = GND; A51 = !e50 & e51 # e50 & !e51 # e50 & e51;

1	A6	A60 = !e60 & !e61 # e60 & e61;
		A61 = !e60 & e61 # e60 & !e61;

1	A7	A70 = !e70 & e71 # e70 & !e71;
		A71 = e70 & e71;

2	B0	B00 = GND; B01 = !A00 & !A01 & A10 & !A11 # A00 & !A01 & !A10 & !A11 # A00 & !A01 & A10 & !A11 # A00 & !A01 & !A10 & A11 # !A00 & A01 & A10 & !A11;

2	B1	B10 = A20 & !A21 & A30 & !A31 # A20 & !A21 & !A30 & A31;
		B11 = !A20 & !A21 & !A30 & !A31 # A20 & !A21 & !A30 & !A31 # !A20 & A21 & !A30 & !A31 # !A20 & A21 & A30 & !A31 # !A20 & A21 & !A30 & A31;

2	B2	B20 = !A40 & !A41 & !A50 & !A51 # !A40 & !A41 & !A50 & A51; B21 = !A40 & !A41 & A50 & !A51 # A40 & !A41 & A50 & !A51 # !A40 & A41 & !A50 & !A51 # !A40 & A41 & A50 & !A51 # !A40 & A41 & !A50 & A51;
2	B3	B30 = A60 & !A61 & A70 & !A71 # !A60 & A61 & A70 & !A71; B31 = !A60 & !A61 & !A70 & !A71 # !A60 & !A61 & A70 & !A71 # !A60 & !A61 & !A70 & A71 # A60 & !A61 & !A70 & A71 # !A60 & A61 & !A70 & A71;
3	C0	C00 = !B00 & !B01 & !B10 & !B11 # !B00 & B01 & !B10 & !B11; C01 = !B00 & !B01 & !B10 & B11 # B00 & !B01 & !B10 & !B11 # B00 & !B01 & B10 & !B11 # B00 & !B01 & !B10 & B11 # !B00 & B01 & !B10 & B11;
3	C1	C10 = B20 & !B21 & !B30 & !B31; C11 = B20 & !B21 & !B30 & B31 # !B20 & B21 & !B30 & !B31;
4	D0	D00 = !C00 & !C01 & C10 & !C11 # !C00 & !C01 & !C10 & C11 # !C00 & !C01 & C10 & !C11; D01 = !C00 & !C01 & !C10 & !C11 # C00 & !C01 & C10 & !C11 # !C00 & C01 & !C10 & C11;

The next step of the neural network development with the FLECHA matrix was the specification of the programmable logic cells that will implement all the 60 neurons. The logic cells can implement any three-input combinatorial function by the representation of its truth table. Each neuron of the pyramid has two inputs (each one represented by two bits) and has two bits as output (to represent the three possible states). Thus, each output bit can be mapped by a 4-input truth table, and a basic neuron needs a group of 6 logic cells to be implemented.

Figure 8 presents these cells.

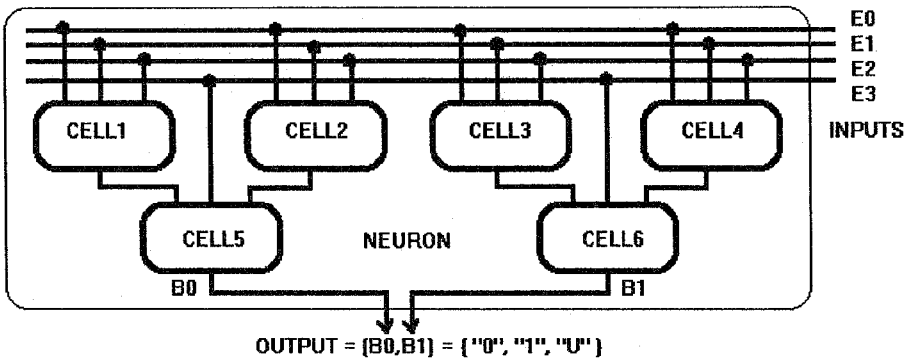

Fig. 8: The implementation of a basic neuron with six logic cells.

The first layer neurons receive two inputs of just one bit, and so, they need two truth tables of two inputs to represent their two bits output. These neurons can be implemented by just two logic cells of the FLECHA matrix. If we join each two neuron of the first layer to its corresponding second layer neuron, it is possible to represent them by just 6 logic cells, since this logic set has only two inputs and one output, all of two bits. The described implementation takes 42 logic cells to represent each pyramid. The maximum delay to propagate signals through a pyramid is 36 ns.

The same data-set, with four pyramids, was implemented in the ALTERA system [11], to provide performance comparisons. The utilization of EPLDs (Erasable Programmable Logic Devices) design tool (MAX+PLUSII) permitted each neuron to be directly introduced by its VHDL description, taking into account basic logic gates (NOT, AND and OR). With this data, the MAX+PLUSII performed the global logic synthesis, the partitioning, the timing analysis and chose the most adequate circuit of the MAX5000 family, as specified by the user. The system defined EPM5032 with 28 pins as the most suitable circuit. From this 28 pins, 11 were used to input and 8 to data output for the four employed pyramids. Just 8 logic cells were used (25% of the total capacity).

In the logic synthesis phase, the network digital circuit suffered a drastic simplification, which permitted the utilization of just two EPM5032 logic cells to calculate the two outputs of each pyramid. Five of the inputs are irrelevant to calculate the pyramid outputs, in this specific application, and they were disregarded by the system. This represents a high redundancy of the ANNs. The logic synthesis and choice of the appropriate circuit were performed in 25 seconds in a 486DX33 MHz platform. The maximum propagation delay in the circuit is 42 ns, representing then a high performance to the EPLD implemented network.

The minimized net was also implemented in the FLECHA matrix. To do so, the MAX+PLUSII output data, obtained by the logic synthesis, was used, since the specific FLECHA matrix tools are under development.

The implemented solution of FLECHA differs from the solution of ALTERA, since cells of FLECHA have individual low capacity, leading to the need of grouping many cells to treat each output. The implementation of the complete neural network used 34 of the 40 available logic cells of the matrix, presenting a maximum delay of 30 ns in the worst case.

The results obtained with the two systems have shown high recognition rates: 95%, with the logic inversion of one bit in the input pattern, and 80%, with two bits presenting logic inversion. In these cases, 100 patterns were used in the training phase. The recognition rates and performances show that this kind of network is suitable to real time neural applications.

5. Conclusions

This work has proved the ability of the GSN model in integrated implementations, specially in FPGAs, since

they presented reduced logic, optimizing the number of necessary logic cells to the implementation. This feature constitutes a big factor of merit in comparison to fixed GSN ANN implementations, as DIANNE. As mentioned, the DIANNE chip must contain all GSN neurons in order to provide flexibility.

The programmable structures of the FPGA allow a network optimization linked directly to the tought patterns, reducing its generality, its implementation costs, and increasing its performance. The complete re-design of the circuit can be made in a short time delay, enabling the circuit to be tested and validated under different conditions imposed to the hardware system.

The design of a direct interface between the GSN software simulator and the FPGA prototyping tools, using the VHDL language, has permitted a significant reduction in the network implementation time. The modification of the output data format, delivered by the GSN simulator to the VHDL language, permits these prototyping tools to recognize directly the constitution of the network neurons, without the need of the user's interventions.

6. References

[1] R. Domínguez-Castro, A. Rodríguez-Vázques, J. L. Huertas and Sánchez-Sinencio, **Analog Neural Programmable Optimizers in CMOS VLSI Technologies**, IEEE Journal of Solid-State Circuits, Vol. 27, No. 7, July 1992, pp. 1110-1115.

[2] C. Lehmann and F. Blayo, **Digital VLSI Generic Elements for Neuro-Emulators using Systolization**, International Workshop on Algorithms and Parallel VLSI Architectures, June 1990, pp. 19-21.

[3] Luís Felipe Uebel and Dante Barone, **DIANNE: A GSN Neural Network VLSI Implementation**, 5th International Symposium on IC Technology, Systems & Applications, September 1993, pp. 678-682.

[4] Luís Felipe Uebel and Dante Barone, **Implementação de uma Rede Neural GSN em um Circuito Integrado**, X Simpósio Brasileiro de Inteligência Artificial, October 1993, pp. 459-468.

[5] E. C. D. B. Filho, **Investigation of Boolean Neural Networks based on a Novel Goal-Seeking Neuron**, Ph.D. tese, University of Kent, England, 1989.

[6] E. C. D. B. Filho, D. L. Bisset and M. C. Fairhurst, **A Goal Seeking Neural for Boolean Neural Networks**, Proc. International Neural Network Conference, Paris, France, Vol. 2, July 1990, pp. 894-897.

[7] E. C. D. B. Filho, M. C. Fairhurst and D. L. Bisset, **Analysis of Saturation Problem in RAM-Based Neural Network**, Electronics Letters, Vol. 28, No. 4, February 1992, pp. 345-346.

[8] I. Aleksander and W. K. Kan, **A Probabilistic Logic Neuron Network for Associative Learning**, Proc. IEEE First Intl. Conf. on Neural Networks, San Diego, California, Vol. II, June 1987, pp. 541-548.

[9] H. Hung-Cheng, K. Dong, J. Y. Ja and R. Kanazawa, **A 9000 - Gate User-Programmable Gate Array**, IEEE Custom Integrated Circuits Conference, 1988, San Jose, California, pp.15.3.1-15.3.7.

[10] Eduardo do Valle Simões and Dante Barone, **Projeto e Implementação de uma Célula Lógica Programável do Tipo Field-Programmable Gate Array**, VII SBCCI - Simpósio Brasileiro de Concepção de Circitos Integrados, Rio de Janeiro, September 1992.

[11] Altera Corporation, **MAX+PLUS II - Getting Started**, 1992, 137 p.