Transactions Briefs

A New Architecture for the Automatic Design of Custom Digital Neural Network

William Fornaciari and Fabio Salice

Abstract—**This brief presents a novel high-performance architecture for implementation of custom digital feed forward neural networks, without on-line learning capabilities. The proposed methodology covers the entire design flow of a neural application, by addressing the internal neuron's structure, the system level organization of the processing elements, the mapping of the abstract neural topology (obtained through simulation) onto the given digital system and eventually the actual synthesis. Experimental results as well as a brief description of the software environment supporting the proposed methodology are also included.**

I. Introduction

The digital solutions for *neurocomputer* implementation proposed by various authors [1]–[8], ranging from dedicated hardware to programmable processors similar to conventional computers, can be roughly classified in two categories: *special-purpose* and *general-purpose* neurocomputer.

General purpose neurocomputers emphasize flexibility in spite of performance and silicon area. For a wide range of purposes, e.g., real-time applications, performance is particularly significant so that the use of special-purpose neurocomputers is required. According to the relationship among neurons (N's) and processing elements (PE's), special-purpose neurocomputers fall into two main classes: *one PE per many (or one) N's* and *one N per many (or one) PE's*. The former is represented by architectures in which the circuit complexity has been reduced by accepting a performance degradation. Typically, the computational cycle of each neuron is not uniform (e.g., as for multilayer feed-forward neural networks with different layer cardinalities) so that some clock cycles have to be used to synchronize neuron activations with the corresponding weights. On the contrary, silicon area used for *active* components is efficiently used since these elements (adders, multipliers$\cdots$) are shared among different neurons. The proposal reported in [4] falls into this class.

The latter class achieves high performance, it consists of architectures in which the neuron functionality is spread over some processing elements concurrently working. The drawback of this approach is that replication of computational blocks to build each single neuron, increases silicon area. This class can be represented by two main digital implementations having different degree of distribution of the neural operations. A first architecture is reported in [1]. The neural computation is realized via a systolic array, avoiding any sort of time multiplexing of synapses onto interconnection lines. This structure is not particularly suitable for feed-forward neural networks with a realistic number of connections because of both the number of processing elements (a multiplier and an adder for each weight) and the complex structure of the synchronization signal.

Manuscript received May 12, 1994; revised October 17, 1994.

W. Fornaciari is with CEFRIEL, Centro per la Ricerca e la Formazione in Ingegneria dell'Informazione del Politecnico di Milano, 20126 Milano, Italy.

F. Salice is with the Dipartimento di Elettronica ed Informazione, Politecnico di Milano, 20133 Milan, Italy.

IEEE Log Number 9415603.

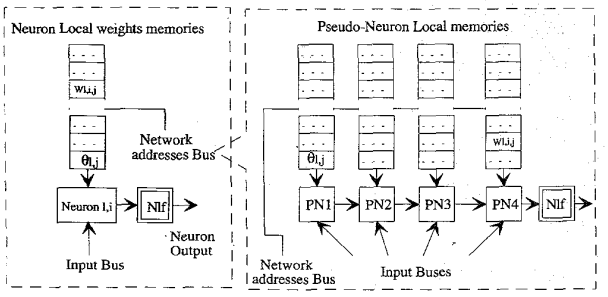

Fig. 1. Decomposition of a neuron onto four pseudoneurons (PN's).

A solution overcoming such a problem has been presented in [9], [10]. The network consists of neuro-cells, called pseudo-neurons (PN's), that can be assembled together in a linear array structure to build a neuron, as shown in Fig. 1. Each PN consists of a local ROM for weights storage and of a unit capable of either evaluating the summation of weighted inputs (added, if necessary, to a similar summation coming from a previous neuro-cell) or forwarding this value to the subsequent elements. In such a solution latency and throughput are improved because of pipelined evaluation of operations and optimization of bus accesses. Furthermore, if all PN's compute the same number of synaptic products, synchronization is simplified and no wait cycle for interlayer data transfer needs to be introduced. Even though this architecture offers a good tradeoff between area and performance, it uses a large amount of delay-cells to synchronize data transfer between both PN's and layers and requires a complex network of synchronization signals.

The goal of this paper is to present a novel digital neuron architecture, still based on a PN approach, named *tree structure*. As it will be shown it obtains lower latency with the same throughput and hardware resources as the linear one. Moreover the performance and area costs due to synchronization signals are greatly reduced; a formal design methodology to fully define the entire neural network structure has been developed. The final hardware description is given in terms of a set of VHDL parametric cells suitable to obtain automatic VLSI implementation through existing CAD synthesis tools.

The next section contains a description of the neuron architecture together with its main properties. Section III reports a graph-based formalization of both computation and structure and an overview of the design methodology. The organization of the design environment implementing the proposed methodology linking existing top-level neural simulator [11] to a VHDL-based synthesis environment, it is also outlined. Section IV presents implementation details and results concerning the neural cells as well as the overall network. Some experimental data have been obtained by implementing a test-case neural network.

II. The Tree Structure

This section introduces a neuron architecture based on the PN approach, suitable for VLSI custom neural network. This architecture achieves much lower *latency* than the *linear array* structure [9] with the same throughput and without any additional hardware requirements.

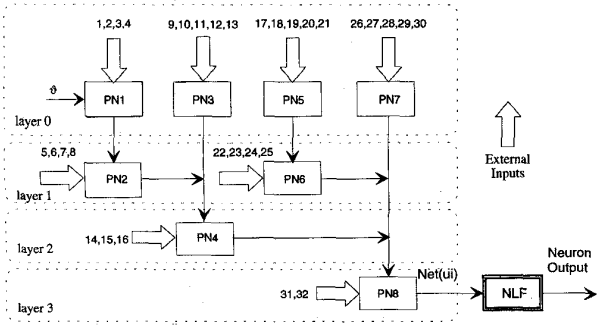

Fig. 2. Internal architecture of a neuron: tree structure of a 32 inputs neuron.

	← PNw →						← ⌈log2(K)⌉ →					
PN1	ϑ	y	1	2	3	4	ϑ	y	1	2	3	4	ϑ
PN2	8	x	y	5	6	7	8	x	y	5	6	7	8
PN3	13	y	9	10	11	12	13	y	9	10	11	12	13
PN4	16	x	x	y	14	15	16	x	x	y	14	15	16
PN5	21	y	17	18	19	20	21	y	17	18	19	20	21
PN6	...	x	y	22	23	24	25	x	y	22	23	24	25
PN7	...	...	26	27	28	29	30	y	26	27	28	29	30
PN8	...	...	...	x	Fire	31	32	x	x	x	Fire	31	32
Time	-1	0	1	2	3	4	5	6	7	8	9	10	11

Fig. 3. Temporal distribution of weights and linear outputs for one neuron composed by a tree of 8 PN. PN's belonging to the same layer forward their linear output at the same time step. In some cases, more than one PN weight positions can be used to collapse partial results from previous layers.

Each PN is only able to add two terms at a time (see Fig. 6): the first can be either a product between neuron input and corresponding weight or a linear input (LI) from another PN, while the second one is a value stored in an accumulator containing, step by step, the partial summation. Let S_i be the weighted inputs summation pertaining to PN_i; to improve the computational parallelism it is possible to arrange the set of partial summations in pairs working together as in the following equation:

$$\sum_{j=1}^{N_{l-1}} (w_{l,i,j}X_{l-1,i}) - \vartheta_{l,i} = (\cdots(((\vartheta + S1) + (S2 + S3)) + ((S4 + S5) + (S6 + S7)))\cdots + (Sn - 1 + Sn)\cdots).$$

Where N_1 is the number of neurons in layer l (in each layer, neurons are numbered from 1 to N_l); $w_{l,i,j}$ synaptic weight value of connection between neuron i of layer l and neuron j of layer $l-1$; $\vartheta_{l,i}$ is the threshold value for neuron i in layer l; $X_{l,i}$ output value of neuron i in layer l; PNw is the number of words stored locally in each PN memory. To obtain such a computational flow, the internal architecture of each neuron is organized as shown in Fig. 2.

The internal structure of the neuron (with K PN's), can by considered as a *tree* composed of $\lfloor \log_2(K) \rfloor$ *internal layers* of PN's, where each PN_i computes its output by adding to S_i a number of LI's produced by PN's belonging to the previous internal layers of the neuron. The number of PN's forwarding their linear inputs to the considered PN_i, depends on the position of PN_i itself within the *tree*. An example of data propagation among different PN's composing a neuron, is shown in Fig. 3. The latency of a tree neuron, i.e. the delay between the process of the first weight and the neuron fire, is $latency_{tree} = \mathrm{PN}w + \lceil \log_2(K_l) \rceil$, where $K_l = \lceil \frac{N_l}{\mathrm{PN}w-2} \rceil$ represents the number of PN's per neuron in layer l.

Each cell of Fig. 3 contains a numerical identifier representing a weight $w_{l,i,j}$ inside the PN memory, while x, y and *Fire* are not important for the computation; they are used only for synchronization, to allow both partial sums propagation through PN's (x and y) and forwarding of the neuron output to the following neurons (*Fire*). The sampling of the partial sum from the previous PN takes place during the time step x while the new partial sum is made available during the y time step.

Concerning the functional equivalence between the tree and the linear array neuron structures, it can be proved the following theorem: *let* PNw *be the* PN's *memory size and* I *the number of neuron inputs. A tree structure dealing with* I *inputs can be realized by using the same number* K *of* PN's *necessary for the linear array architecture.* $I = K(\mathrm{PN}w - 2)$. Although a formal proof can be found in [12], it can be justified by considering that both linear array and tree structures are composed of K PN's (with the same memory size PNw), so that 2 K memory words are used for collapsing of the partial summation and firing. As a consequence, the number of memory locations available for storing the synaptic weights is the same for both structures.

A significant difference between tree and linear array is in term of performance. As far as latency is concerned, the use of a tree structure allows to obtain an improvement with respect to the one of the linear array, that is $\mathrm{PN}w + K - 1$, due to an optimized collapsing of the LI's produced by the various PN's. The improvement depends almost-linearly on K and evaluates

$$\text{Latency gain} = latency_{\text{linear array}} - latency_{\text{tree}} = (K-1) - \lceil \log_2(K) \rceil$$

The actual gain starts with $K > 3$ because for $K < 4$ both structures correspond to the same connection of PN's. Furthermore, the PN's memory size (PNw) has to be sufficient to make possible the collapsing of the maximum number of LI pertaining to a single PN of the *tree*. Hence, it is necessary to fulfill the feasibility constraint of having $\mathrm{PN}w \geq \lfloor \log_2(K) \rfloor + 1$.

Both structures have the same throughput $f_{ck}/\mathrm{PN}w$ since:

- given a neural network topology, the number and the memory size of each PN is the same;
- they use the same type of PN, i.e., they have the same clock and overall computational cycle equal to the number of memory words to be processed.

As it will be shown in Section IV, a proper coding of the tree structure memory contents will also allow to avoid any broadcast of signals to control the PN computation, i.e., to relax the connectivity requirements of the digital implementation.

III. The Design Methodology

The design methodology is mainly based upon an internal structure representation of the PN-based neuron (both *linear array* or *tree*) by means of a direct graph (digraph) [13]. For such a type of modeling, we obtained some formal rules which allow to determine the position of all weights within each PN memory (both positions x, y used for synchronization and synaptic weights). This representation constitutes the basic support of our approach to ANN design since the digital architecture is automatically obtained by exploiting some of the digraph model properties.

Each *node* of the neuron digraph modeling corresponds to one of its PN's while each oriented arc represents the data transfer between PN's (see Fig. 4).

The *digraph* model can be formally described by its *adjacency matrix* $A = [a_{ij}] - K \times K$-, where $a_{ij} = 1$ if there is an *arc* from PN_i to PN_j, otherwise $a_{ij} = 0$. If a *linear array* architecture is considered, the *adjacency* matrix has only the first upper codiagonal composed of 1, because each element i is connected to element $i+1$. For a *tree* structure composed of K PN's, the non-null elements $a_{i,j}$ of A are those satisfying the following conditions: $i = 2^t s + 2^{t-1}$

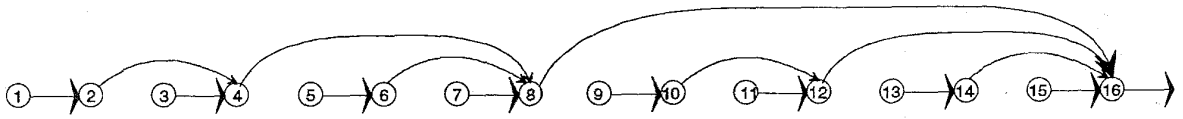

Fig. 4. Digraph representation of a neuron composed of 16 PN's in a tree configuration.

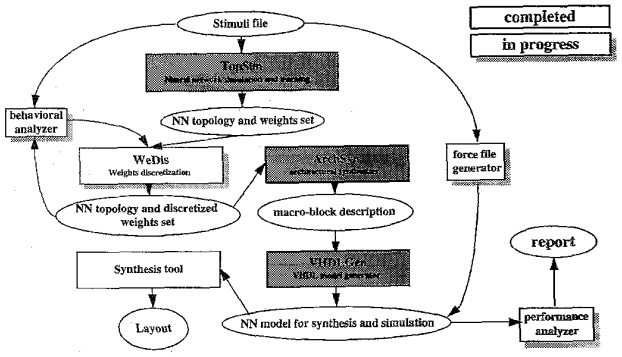

Fig. 5. The complete design flow.

and $j = \min((2^t s + 2^t), K)$, with t ranging in $[1, \lceil \log_2(K) \rceil]$ and s ranging in $[0, \max((\lfloor K/2^t \rfloor - 1), 0)]$.

In fact, if $t = 1$ and for each value of s, the elements of A set to 1 represent links from the nodes of the *basis* (nodes with indegree = 0) to elements at distance 1 from them; if $t = 2$ and for each value of s the elements of A equal to 1 corresponds to the links from the previous nodes (at distance 1 from the *basis*) to nodes at distance 1 from them, and so on for the other values of t. Not to exceed the $K \times K$ boundaries of the matrix A, the min/max constraints have been introduced on the j and s ranges in the previous conditions.

It has been proved [13] that each entry of $a^n_{i,j}$ of $A^n = [a^n_{i,j}]$, is the number of *walks* of length n from node i (PN_i) to node j ($\mathrm{PN}j$). Since there exists at the most one path connecting two different nodes, i.e., the digraphs are *in-tree* [13], the elements of A^n can be either 0 or 1.

By exploiting the above structure of the matrix A^n it is possible to efficiently find out the length of the longest path connecting all the possible pairs of PN's. This values are gathered in the longest path matrix LPM, defined as

$$\mathrm{LPM} = \sum_{n=1}^{\bar{n}} n * A^n$$

where

$$\bar{n} = \begin{cases} \lceil \log_2(K) \rceil \text{for a } \textit{tree} \text{ structure} \\ K \text{ for a } \textit{linear array} \end{cases}$$

The longest distance between nodes of our digraph model is used in our methodology to represent the computational delays which have to be introduced to guarantee a proper synchronization of the computation among PN's.

The final formal representation produced by the methodology is obtained starting from the network topology (the layers number and the cardinality of neurons per layer), the weight sets of each neuron and the architectural parameters (either K or PN and the type of neuron: *tree* or *linear array*). It is summarized by the following information

- allocation of *synaptic weights* within the PN memories (it is stored in the final network matrix, **MRF**);
- definition of the *delay cells* required to obtain synchronization both inside the layer and between adjacent layers (it is represented by the set of delay number tables, $\{\mathbf{NTD_l}\}$);

From this information the complete digital architecture is established; in fact, positioning of the delay cells and allocation of the synaptic weights within the PN memories allow the mapping of interconnections onto the buses, and guarantee synchronization both inside the neuron and among all neurons. The methodology for obtaining MRF and the set of $\mathrm{NTD}ls$ is based on the following sequence of steps:

Step 1. Guarantee a correct synchronization *within each neuron*; this condition is expressed by the neuron number table $\mathbf{NTN_{l,i}}$, containing the synaptic weights of neuron i of the layer j, so that correct synchronization is obtained.

Step 2. Guarantee the synchronization *between adjacent layers*, by extracting the delay number tables $\mathbf{NTD_l}$s, each defining for its layer l, how the input signals to the various PN's must be delayed.

Step 3. Obtain the correct placement of weights *inside the layer* by collapsing all the $\mathbf{NTN_{l,i}}$ of layer l into the layer number table, $\mathbf{NTL_l}$, of layer l.

Step 4. Derive from $\mathbf{NTL_l}$ the network number table **NTR**, to represent the correct weights synchronization for the *whole neural network*.

Step 5. Extract the relative position of *all the weights during a computational cycle*, namely the placement of weights in the PN's memories. The result of this step is the matrix **MRF**.

The description of the complete algorithm for the automatic synthesis as well as examples of its application are given in [14]. The final formal description of the VLSI macrocell architecture, is eventually automatically translated in a corresponding VHDL code for synthesis. In fact, this design methodology and the related tool are part of a more general design environment starting from an abstract view of the problem (see Fig. 5).

The first stage of the design flow concerning a neural application is the functional simulation and the learning process in order to find out both the weights set and the network topology able to solve the given problem (*TopSim*). Such an abstract description is mapped onto an optimized digital structure (*tree* or *linear array*) by using the previously discussed methodology implemented in *ArchSyn*. Before committing the optimized macroblock description, *WeDis* performs a preliminary processing to find out, by assuming the learning error as a constraint, both a discretized weights set and the number of steps required by each approximated nonlinear activation function. The obtained macrocell system implementation will be converted into a VHDL description by *VHDLGen*. The VHDL network model is useful as input for both final layout synthesis tools and process analyzer. The software system has been developed on a SUN SPARCStation II; VHDL simulation and synthesis use the View Logic POWER VIEW and SILC- SYN 3 tools.

IV. Implementation and Experimental Results

A suitable encoding of the PN memory words has been adopted both to store the synaptic weights and to locally generate the control signals for the data transfer. Such a memory encoding allows to reduce the connectivity requirement avoiding any broadcast of control

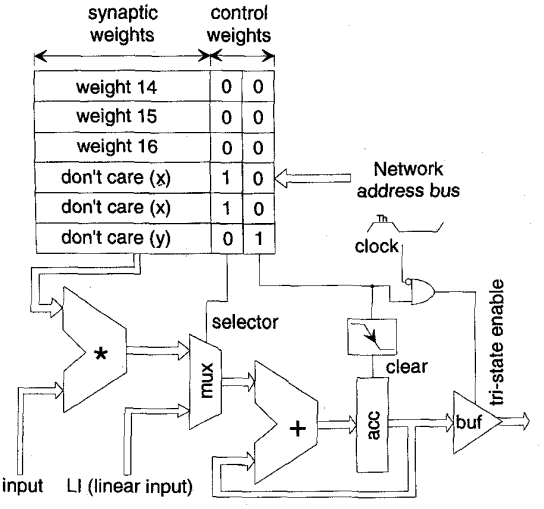

Fig. 6. Format and usage of the memory words stored in PN4 local memory of the neuron described in Fig. 3.

signals among PN's. The words locally stored in each PN are composed of two parts (see Fig. 6):

- *a weight block*; consisting of a certain number of bits used to store the connection weight;
- *a synchronization block*, that is a control-part composed of two bits. The first allows to realize the multiplexing of PN's onto both inter-PN's connection lines and inter-layer bus by controlling their three-state buffered outputs (value 1 enables the output); the corresponding weights are both y and *fire* (see Fig. 6). The second is used, internally to each PN, to select either the synaptic product (value 0) or the LI (value 1) as the adder input (see Fig. 6); the corresponding weight is x.

Referring to the weight block, it has to be pointed out that the words not used to store synaptic weights (the ones marked as x and y in Fig. 6) can be considered as *don't care* bits set so that it is possible to obtain a reduction in the memory area if a PLA solution is adopted. In addition, another connectivity reduction can be obtained for the tree structure due to not uniform distribution of weight among PN memories. In fact, for the *linear array*, the number of buses connecting the layer $l-1$ to the layer l is equal to the number of PN's composing the neurons of layer l (i.e., K_l), while whenever the last PN of the *tree* structure is used only to collapse partial computations, its input set is empty so that no bus line is needed (memory word are used only for synchronization purposes).

In order to match the requirements of computation process ordering of the first hidden layer, the input signals belonging to the environment have to be correctly sampled. To support such a sampling process, an interface circuit is designed by using some information, implicitly included in MRF and $\{\mathrm{NTD}_l\}$, related to the hidden layer synchronization.

The interface circuit presents two classes of signals: *input* and *synchronization*. The first class is partitioned into some sets, each containing inputs using the same bus, that are completely defined by $\mathrm{NTD}_{\mathrm{first\_hidden\_layer}}$. Each signal of the second class defines both the temporal points in which the input signals can be changed (level 0) and the temporal period in which the input signals have to be stable (level 1). The sampling of inputs set is performed on the falling edge. The definition of the above temporal periods is completely specified by the position of entries y and *fire* inside the sub-matrix of MRF corresponding to the first hidden layer. An example of input interface circuit is reported in Fig. 7.

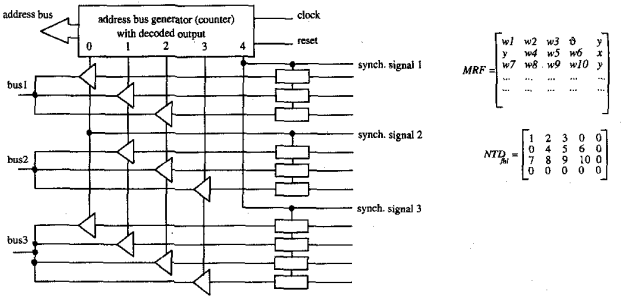

Fig. 7. Example of input interface.

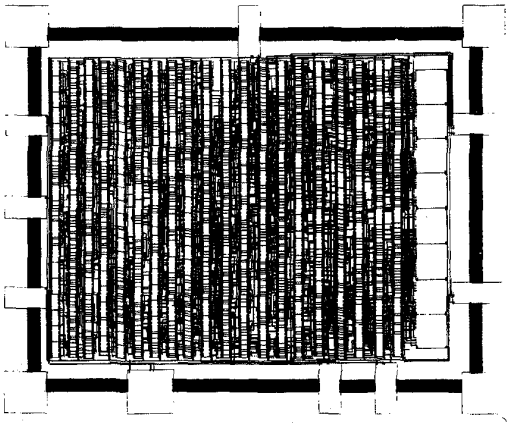

Fig. 8. The chip layout of the neural network implementation.

To validate both architecture and methodology, some experimental results have been obtained by implementing a test-case neural network composed of 8 neurons with 6 inputs, 4 neurons in the hidden layer and 4 output neurons. The implemented nonlinearity is a 2's power stepwise approximation of the *sigmoidal* function because of its suitability for VLSI implementation [15]. In particular, one and three steps functions are respectively employed for the hidden and the output layers. In such a way it is possible to simplify the multipliers that, for three steps nonlinear functions, leads to the complexity of 4 equivalent-gates per bit. The weights are encoded on 8 bits while the summation units are realized as ripple carry structures. The project has been developed at SGS-THOMSON by using a standard cells 1.5 μ double metal process (the layout is reported in Fig. 8.).

The chip, whose layout is core-limited, also includes the logic for the cyclic generation of the memory bus address which is broadcasted to all PN's memory. The memories, one for each PN, have been implemented through PLA's. The computational part of each PN takes about 0.3 mm^2 while the PLA's take less than 10% of the global area which is, pad included, 4227×3496 μ^2. The clock frequency is approximately 250 KHz and the circuit performance are reported in Fig. 9.

Further results have been obtained by producing a gate-level description of a neural network with 25 inputs of 8 bit each, 20 hidden neurons and 1 output neuron. They use a stepwise approximation of the nonlinear function, with one and nine levels respectively employed for output and hidden layers; weights have been represented onto 12 bits. Two different implementations (NN_1 and NN_2) have been produced by using respectively the cost functions $area^{0.5} * latency^{0.5}$ and $area^0 * latency^1$ to drive the decision concerning the architectural parameters. The optimal PNws pertaining such

Latency/Clock frequency: ..... 44 μsec	Throughput: 41 KHz
Clock frequency: ..... 250 KHz	Routing area 3.2 mm^2 (36%)
Global area (pad included) ..... 4227 μ x 3496 μ	Core area (PLAs + Routing + PNs) 3396 μ x 2622 μ
PN area ..... 0.3 mm^2	PLAs area 338 μ x 2622 μ=0.88mm^2

Fig. 9. Circuit performance and final layout data.

	NN_1	NN_2	
PLAs size (*PNw*)	11	7	
PNs per *hidden layer* neuron	3	3	~1 mm^2 each
PNs per *output layer* neuron	3	4	~0.5 mm^2 each
Total number of PNs	60+3	100+4	
Number of gates	120886	197380	
Global area [1.5μm] (PLAs+PNs+routing)	~65 mm^2		~100 mm^2
Latency/Clock frequency	2.76 μsec	2.18 μsec	
Throughput (f_{ck}/*PNw*)	0.79 MHz	1.24 MHz	
Clock frequency	8.7 MHz	8.7 MHz	

Fig. 10. Circuits characterization.

networks, obtained by means of *TopSim*, are respectively 11 for the former and 7 for the latter. Fig. 10 reports data on area and performance.

V. Conclusion and Future Developments

A neuron model and a formal methodology and a neuron model for obtaining the complete architecture of a feed-forward neural network for fully dedicated systems without on-line learning capabilities, have been presented. The methodology allows the creation of the network by mapping each neuron onto some basic processing elements connected either as a *tree* or a *linear array* structure. The final result is the complete description of the neural network, namely the organization of PN's inside each neuron, the bussed interconnection structure among neurons and the positioning of delay cells such that synchronization between layers is guaranteed. With respect to other proposals of the literature, this solution is characterized by better latency and throughput, together with a more efficient connection strategy and the possibility of fully automating the whole design process. The description of the digital architecture consists of some matrixes that can be simply arranged for the final VLSI synthesis. This approach is particularly suitable to support automatic synthesis based upon a library of building blocks (e.g. PN's and delay cells). The methodology has been tested and experimented by realizing a design environment running on SUN SPARC station 2 and by implementing some test-case neural networks passing through an intermediate VHDL description. The data collected show that the proposed architecture is characterized by good performances while efficiently fulfilling the connection requirement imposed by the application. Currently the system has been improved to cover also the Hopfield model. Further work will be in the direction of supporting a wider range of neural models through extension of the macrocells library and of the interconnection capacity.

References

[1] P. Ienne and M. Viredaz, "GENESIS IV: A bit-serial processing element for a multi-model neural network accelerator," in *Proc., Appli. Specific Array Processors '93*, Venice, Italy, Oct. 1993, pp. 345–356.

[2] U. Ramacher, A. Raab, J. Anlauf, U. Hachmann, and M. Wesseling, "SYNAPSE-X: A general-purpose neurocomputer," in *Proc., IEEE-IFIP MicroNeuro-91*, Munich, Germany, Oct. 1991, pp. 401–409.

[3] S. Jones, M. Thomaz, and K. Sammut, "Linear systolic neural network engine," in *Proc, IFIP Workshop on Parallel Architectures on Silicon*, Grenoble, France, Dec. 1989.

[4] S. Y. Kung and J. N. Hwang, "A unified systolic architecture for artificial neural networks," *J. Parallel Distrib. Comput.*, 1989.

[5] P. Y. Alla, G. Dreyfus, J. D. Gascuel, A. Johnnet, L. Personnaz, J. Roman, and M. Weinfeld, "Silicon integration of learning algorithm and other auto-adaptative properties in a digital feedback neural network," in *Proc., IEEE-ITG Workshop on Microelectronics for Neural Networks*, Dortmund, Germany, 1990.

[6] J. G. Delgado-Frias, S. Vassiliadis, G. G. Pechanek, and W. Lin, "A VLSI pipelined neuroemulator," in *VLSI for Neural Networks and Artificial Intelligence*, J. G. Delgado-Frias and W. R. Moore Eds. New York: Plenum, 1994, pp. 71–80.

[7] J. Ouali and G. Saucier, "Fast generation of neuron ASIC's," in *Proc. Int. Joint Conf. Neural Networks, IJCNN'90*, San Diego, CA, June 17–21, 1990.

[8] P. Treleaven and M. Vellasco, "Neural networks on silicon," in *Proc. IFIP 3rd Workshop Wafer Scale Integration*, Como, Italy, June 1989.

[9] F. Distante, M. G. Sami, R. Stefanelli, and G. S. Gajani, "A compact and fast silicon implementation for layered neural nets," *VLSI for Artificial Intelligence and Neural Networks*, J. G. Delgado Frias and W. R. Moore Eds. New York: Plenum, 1991.

[10] W. Fornaciari, F. Salice, and G. Storti Gajani, "A formal methodology for automatic synthesis of neural networks," in *Proc., IEEE-IFIP MicroNeuro-91*, Munich, Germany, Oct. 1991.

[11] *Neural Works II Reference and User Manuals*, NeuralWare Inc., 1989.

[12] W. Fornaciari and F. Salice, "A low latency digital neural network architecture," in *VLSI for Neural Networks and Artificial Intelligence*, J. G. Delgado Frias and W. R. Moore Eds.New York: Plenum, 1994, pp. 71–91.

[13] F. Haray, *Graph Theory*. Menlo Park, CA: Addison-Wesley, 1969.

[14] W. Fornaciari and F. Salice, "A structured approach for automatic design of PN-based digital neural networks," in *Proc., IEEE-IJCNN '92*, Beijing, China, Nov. 1992.

[15] C. Alippi and G. S. Gajani, "Simple approximation of sigmoidal function: realistic design of neural networks capable of learning," in *Proc., Intern. Symp. Circuits Syst.'91*, Singapore, June 1991.