A Fast, Configurable Digital Neural Network ASIC for High-Speed Applications*

José Luis Ayala, Antonio G. Lomeña, María Luisa López-Vallejo and **Ángel Fernández**
Departamento de Ingeniería Electrónica
Universidad Politécnica de Madrid, Madrid (Spain)
e-mail: {jayala,lomena,marisa,angelfh}@die.upm.es

Abstract

We present a hardware solution that optimally implements several configurations of Neural Networks and has the possibility of on-chip learning. This architecture provides not only the efficiency of the hardware realizations but also the adaptability and flexibility of the software implementations. This solution uses an efficient pipeline in order to improve the throughput of the system in real time applications. Thanks to a novel implementation of the sigmoid activation function (through the modification of the A-Law, based on a completely combinational module), the system devised has neither the area restrictions of traditional solutions using internal LUTs nor the time delays involved in external memories access.

1 Introduction

Although modern computers are very competent doing repetitive and numerically intensive tasks, they are not so efficient solving other tasks which appear simple for humans. In that sense, humans are able to learn and remember new concepts, techniques and methods with the only help of examples and a little feedback of an *instructor*. Humans have also the ability to extract complex rules from their decisions and reasonings [4].

Artificial Neural Networks are inspired in biological nervous systems; they try to emulate the functionality and architecture of the processing units (neurons) and the topology of these systems (synapses and layers) looking for the main functions of the reasoning, the remembering and the generalization. However, these mathematical models must perform a great number of parallel computations, all of them expensive in terms of CPU cycles. We can conclude that even though software simulation is the most immediate and easy solution, it is not the best choice for high rate applications.

Neural Networks begin to be used in a large range of commercial applications, all of them with a set of well defined necessities: they need a real time response from the net (especially during the recall phase), they have to develop a great number of operations, once the topology of the net is defined for the application itself, it is not usually changed; and, finally, they will be necessary in new portable devices. All these requirements force us to use a hardware solution instead of the traditional software platforms. However, traditional analog electronics is not able to bring a solution with enough precision and reconfigurability; and digital programmable devices such as FPGAs and DSPs have not enough integrated logic capacity, thus forcing us to use several of these components.

The architecture we are introducing is an ASIC capable of on-chip learning providing fast response with high reconfigurability. However, one of our most important contributions is the employment of a modified A-Law [12] to implement the activation function. This technique reduces the chip size with respect to traditional LUT based implementations. Besides, the design of the chip, presenting processing units with internal and independent memories, takes into account the possibility of a future interface which could transfer the weights from a PC to the net (allowing the possibility of using different learning algorithms).

The paper is organized as follows: section 2 reviews the state-of-the-art in hardware Neural Networks technology. Section 3 shows the fundamentals of Neural Networks theory. In section 4 both the global architecture of the circuit and its main modules are described. Section 5 explains the details concerning the pipeline which has been designed for the system, while section 6 deals with aspects related to the synthesis of the circuit. Finally, section 7 shows the conclusions and future work.

2 Related work

Various hardware implementations of neural networks can be found in the literature. Both analog and digital solutions have been proposed. In most cases, however, these designs reach optimal speed and area results only when they are well suited to an specific topology of the net.

For example, in [10] and [11] an analog VLSI architecture is developed. This solution contains 1024 neurosynapses in an active area of 6.1 mm x 3.6 mm (with 27000 transistors and 1024 MOS capacitors); however, it is very limited by the number of neurons (between 10 and 16). Besides, the reconfigurability of the analog solutions is somewhat poor; if the chip is composed of a certain number of neurons, it will not easily be able to work with nets with a different number of units.

Solutions regarding the implementation of Neural Networks on FPGAs are also abundant, like the one proposed in [3], where it is necessary to introduce many devices (22 FPGAs for a 5:4:2 net) due to the reduced logic integration of the FPGAs used (XC3090). It could be thought that newer models of

* This work is funded by CICYT project +TIC2000-0583-C02-02

these devices, with higher logic integration, could improve these solutions, but it would still remain a slow performance due to FPGA restrictions. However, important advances have been done on this matter as the research of [5].

References can also be found about implementations on DSPs. Usually, Neural Nets have to be implemented with several DSPs ([8] employs 15 devices), because logic integration restrictions of DSPs are similar to FPGA restrictions; nevertheless they can achieve a faster response.

3 Fundamentals of Neural Networks

3.1 Functioning of Neural Nets

An artificial Neural Network is a processing algorithm which shares certain analogies with biological neural nets. Artificial Neural Networks have been developed with the following assumptions:

- Information processing takes place at processing units called *neurons.*
- Signals between neurons are transmitted through connections called *synapses.*
- Each connection has an associated weight, which multiplies the transmitted signal value.
- Each neuron has an associated *activation function* which takes as input the weighted sum of signals to the input of the neuron, and gives the output of the processing unit.

A typical Neural Network is characterized by (1) the distribution of synapses and neurons (*topology*), (2) the algorithm used for finding the weights of the connections (*learning algorithm*), and (3) the activation function of the processing units.

Neural Networks are disposed in *layers.* Neurons belonging to the same layer work in a similar way; so, they usually develop the same activation function and are connected to the same neurons of the following and previous layer. Figure 1 presents a net with two layers[1] and the notation followed to the end of the article (x_i is the input signal to the net, y_k is the output signal and t_k is the target value).

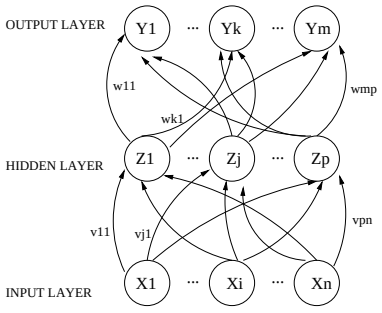

Figure 1: Generalized Network

The functioning of the artificial neuron tries to imitate the behavior of the biological neuron; therefore, if we consider a processing unit like the one presented in figure 2, the output of the unit is:

$$y_{in} = \sum_{i=1}^{n} x_i \cdot w_i \quad (1)$$

$$y = f(y_{in}) \quad (2)$$

where f is the **activation function** of the neuron, typically the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$.

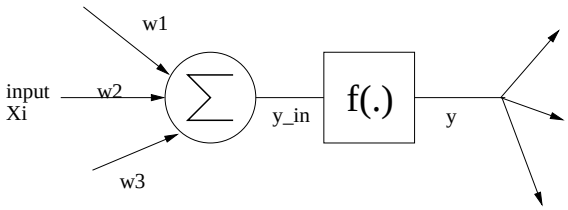

Figure 2: Artificial neuron

The *learning algorithm* is an iterative method used to assign correct values to the weights of the net in order to solve the problem for which the network has been designed. Two types of learning algorithms can be distinguished: *supervised* and *unsupervised.* The main difference between them is that in supervised learning the training vectors are presented with their associated targets, but in unsupervised learning the associated outputs for the training vectors are unknown. Supervised learning algorithms can be used in most minimization, classification and clustering problems; the backpropagation method, a supervised learning algorithm, is one of the most powerful and flexible techniques for training the net [4]; therefore, it has been chosen for our hardware implementation. Next section describes its fundamentals.

3.2 Backpropagation algorithm

Learning algorithm:

1. Initialization of weights with values near to zero
2. While stopping condition is false,
   For each pair of training vectors,
   Forwardpropagation of inputs
   (a) Each input neuron $(X_i, i = 1...n)$ gets its input signal x_i and broadcasts it to all neurons on the hidden layer.
   (b) Each hidden neuron $(Z_j, j = 1...p)$ adds its weighted input signals,

$$z_j^* = \sum_{i=1}^{n} x_i v_{ji} \quad (3)$$

   applies its *activation function* (which generates a useful nonlinear partition of the sample space) to compute its output signal,

$$z_j = f(z_j^*) \quad (4)$$

   and broadcasts this one to all output neurons.
   (c) Each output neuron $(Y_k, k = 1...m)$ adds its weighted input signals,

$$y_k^* = \sum_{j=1}^{p} z_j w_{kj} \quad (5)$$

[1] A two layer topology can solve almost all of the interesting problems [14]

and applies its a*ctivation function* to compute its output signal,

$$y_k = f(y_k^*) \tag{6}$$

Backpropagation of the error

(d) Each output neuron $(Y_k, k = 1...m)$ receives a target (t_k) associated with the training vector, and calculates its information error term,

$$\delta_k = (t_k - y_k) f'(y_k^*) \tag{7}$$

(where f' is the derivative form of f, calculated as $f(y_k^*) \cdot (1 - f(y_k^*))$; it then computes its weight correction term

$$\Delta w_{kj} = \alpha \delta_k z_j \tag{8}$$

(where α, known as *learning factor*, controls the learning speed) and sends δ_k to the hidden neurons.

(e) Each hidden neuron $(Z_j, j = 1...p)$ adds its weighted input deltas,

$$\delta_j^* = \sum_{k=1}^{m} \delta_k w_{kj} \tag{9}$$

multiplies it by the derivative of the activation function in order to calculate its information error term,

$$\delta_j = \delta_j^* f'(z_j^*) \tag{10}$$

and computes the weight correction term

$$\Delta v_{ji} = \alpha \delta_j x_i \tag{11}$$

Weight updating:

(f) Each output neuron $(Y_k, k = 1...m)$ updates its weights $(j = 0...p)$:

$$w_{kj}(new) = w_{kj}(old) + \Delta w_{kj} \tag{12}$$

Each hidden neuron $(Z_j, j = 1...p)$ updates its weights $(i = 0...n)$:

$$v_{ji}(new) = v_{ji}(old) + \Delta v_{ji} \tag{13}$$

(g) Test stopping condition (such as iteration number, error decrease, error stabilization...).

4 Architecture of the circuit

4.1 General architecture

The processing units of the chip are disposed in a ring architecture, having their own internal memories (data and weights) and sharing a set of RAMs (auxiliary memories). This configuration is derived from the classical 1D vector - matrix multiplier, providing an optimal area-speed trade-off, and high reconfigurability. Hence, the system can execute different network configurations. Also, it can achieve high learning and recall speeds and the active area of the chip fits the ASIC standards [6]. Figure 3 shows this general scheme.

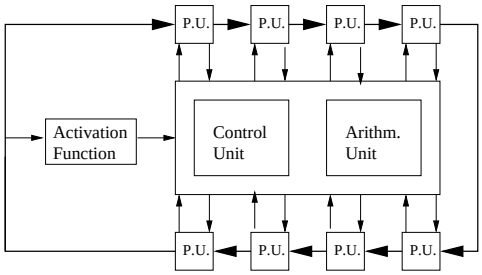

Figure 3: System architecture scheme

During the forwardpropagation phase, the ring carries the partial operations (weighted sums, equations 3-6) from one processing unit to the following. During the backpropagation phase, each processing unit is accessed individually by the control unit, performing the weight updating (equations 12-13).

The architecture we are introducing, and for which we will give the implementation details and results, has eight *processing units* and, due to area restrictions, it is able to implement nets with up to 200 neurons per layer. With this configuration and restriction, the internal RAMs of the processing units must store up to $\frac{200}{8}$ values, giving a reasonable total memory area. These architectural characteristics could be easily changed through the use of implementation parameters.

The weights of a Neural Network can experiment a great variability. There is no algorithm that can predict this weight oscillation, and it depends on the network topology and input data. However, when data are normalized, the synaptic weights exhibit a statistic distribution with high probability around zero and very low probability above 10. This behavior can be corroborated by experiments with a software simulator like SNNS [9]. On the other hand, in order to achieve the highest learning speed, data have to be represented with high accuracy (software simulators usually employ 5 or 6 decimal digits). Due to these restrictions and characteristics, the system has been designed for working with floating point operands, and all the arithmetic subsystems have been devised for achieving such functionality.

An 18 bits mantissa and 6 bits exponent configuration has been chosen, obtaining sufficient resolution and flexibility in the weight oscillation for our purposes. Hence, data near zero can be operated with up to 5 decimal digits. Besides, the range representation of data can be extremely extended but the precision achieved is then reduced.

4.2 Processing units

Each processing unit is composed of a floating point multiplier, a floating point adder, a register, two 32 word RAMs and a set of multiplexers which allow the modification of the connections in order to get the reconfiguration of the units. These can operate as independent neurons, but this is an unefficient manner because the net topology would be limited by the number of hardware neurons and these could never operate as different neurons of the same

layer. Figures 4, 5 and 6 show the reconfiguration of the processing units during the forwardpropagation (equations 1 and 2), the delta computation (equation 9) and the weight updating (equations 12 and 13) phases.

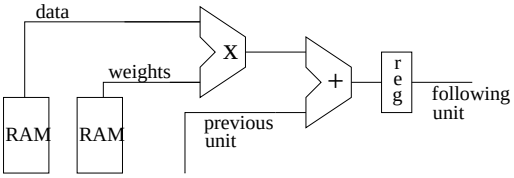

Figure 4: Processing unit during forwardpropagation

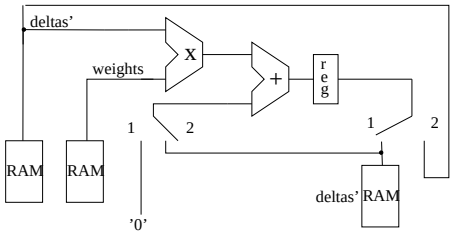

Figure 5: Processing unit during delta computation

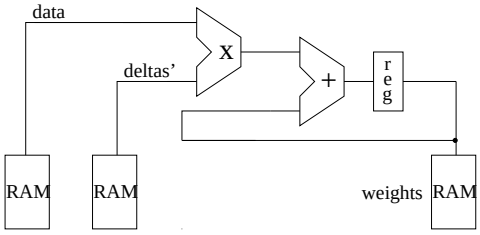

Figure 6: Processing unit during weight updating

The internal memories of each processing unit store the synaptic weights and input data. The data operated by the adder and the multiplier depends on the working phase, and these can be provided from the previous processing unit in the ring, from an external memory or from the processing unit itself.

4.3 Activation function

As it was shown in section 3.2, the activation function is the nonlinear application the neuron uses to generate a useful partition of the sample space. Many functions can be used (linear, gaussian, sinusoidal...), but the most common choice is the sigmoid function $y = \frac{1}{1+e^{-x}}$ because of its excellent characteristics of convergence speed and easy derivability. Traditional hardware implementations of Neural Networks usually employ look-up tables in order to generate the correct values of the function. However, the precision required in the sampling of the function forces to use external memories due to the size of these devices (for example, if 24 bits are used in data codification, a 16 MB LUT would be needed for the sigmoid function tabulation). However, the option of external memories, should be avoided because of the delays introduced in the communication interface. The size of the memories could be reduced employing zero-order and first-order approximations of the sigmoid function, but a huge area in memory devices would be required.

We have developed a combinational approximation to the sigmoid function through the modification of the European compression standard (A-Law) due to the previous considerations. The A-Law of compression is the European standard (CCITT) for approximating the logarithmic compression of voice samples coded with 16 bits [12]. This compressor can be expressed as:

$$y = C(x) = \begin{cases} y_{max} \cdot \frac{A(|x|/x_{max})}{1+\ln(A)} & 0 < \frac{|x|}{x_{max}} < \frac{1}{A} \\ y_{max} \cdot \frac{\ln[A(|x|/x_{max})]}{1+\ln(A)} & \frac{1}{A} < \frac{|x|}{x_{max}} < 1 \end{cases}$$

The A-Law compression scheme approximates that output characteristic with linear segments. Therefore, an output characteristic very similar to the first-order approximation of the sigmoid function is obtained. The main advantage of this approximation is its easiness of implementation with combinational logic, which avoids the large area requirements of other solutions.

The activation function approximation module works with the mantissa of the input data. It generates the output by assigning a segment label to the most representative bits of the operand as it is shown in figure 7. The classification of the operand in the segment it belongs to, takes place with a set of combinational operations; the rest of operations can also be done with combinational logic. Finally, a constant value for the exponent is assigned in order to normalize the output from -0.5 to 0.5. Figure 8 summarizes the fields involved in output data codification.

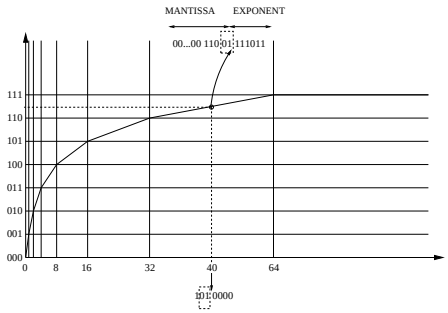

Figure 7: Linear approximation of the sigmoid function

ZEROES	SEGMENT LABEL	M. SIGNIFICANT BITS	CONSTANT EXPONENT (in two's complement)
0000....000	110	01	111011

Figure 8: Data output fields

4.4 Pseudo-random generator

It has been demonstrated [14] that the only requirement for the first initialization of the weights is to choose random values near zero. The chip has been equipped with a simple sequential module capable of generating pseudo-random values near the origin. It is based on a shift register and a logic XOR function of its contents. The mantissa of the output result is formed with the value contained in the shift register while the exponent is adjusted to obtain near zero result data.

Table 1: Throughput

Phase	Clock cycles
Recall	$m + \pi(m) \cdot 8 \cdot \frac{n}{m} + \pi(m) \cdot \frac{n}{m} + n + \pi(n) \cdot 8 \cdot \frac{p}{n} + \pi(n) \cdot \frac{p}{n}$
Learning	$m + \pi(m) \cdot 8 \cdot \frac{n}{m} + \pi(m) \cdot \frac{n}{m} + n + \pi(n) \cdot 8 \cdot \frac{p}{n} + 4 + +n + \pi(n) \cdot \frac{p}{n} + n + n + 2 \cdot \pi(n) \cdot \frac{p}{n}$

4.5 Arithmetic and control unit

The central processing unit of the chip is composed of an arithmetic unit and a control unit. The arithmetic unit develops the operations involved in the weight correction (equations 3-8, 10 and 11) and memories updating (output data from previous layer and weight updating). As has been said in previous sections, all the calculations needed are computed using floating point arithmetic.

On the other hand, the control unit sequenciates the operations of data propagation and orders reconfiguration of the processing units (see section 4.2) by means of several independent finite state machines which operate in a concurrent manner.

5 Pipeline

The system architecture that has been presented in the previous paragraphs performs the parallel processing of the data. However, this functionality would not be possible without a careful disposition of values (weights and inputs/outputs) in the internal memories of the processing units. It is also necessary a mechanism for writing and reading these values [13]. The data transmission from one processing unit to the following in the ring is controlled by an efficient pipeline which operates during the recall and learning phases.

Figures 9 and 10 show the correct disposition of weights and input data for a 2:5:1 net with 4 processing units[2], where each column of the table corresponds to the internal RAM of a processing unit. Reading of values begins from the bottom of the table, one line at each clock cycle, and goes up. Initial values (shaded in the figure) are used for pipeline filling. Once these previous data have been operated (pipeline filled), the system provides one result per clock cycle if the number of neurons per layer is less than or equal to the number of processing units. Otherwise, each result will be obtained in an integer multiple of the clock cycle (equal to the max number of neurons per layer / number of processing units). Also, non-operation cycles will have to be inserted.

Referring to figures 9 and 10, the outputs of the processing units are:

first clock cycle: $x_1 \cdot v_{11}$, 0, 0, 0

second clock cycle: $x_1 \cdot v_{21}$,$x_1 \cdot v_{11} + x_2 \cdot v_{12}$, 0, 0

third clock cycle: $x_1 \cdot v_{31}$, $x_1 \cdot v_{21} + x_2 \cdot v_{22}$, $x_1 \cdot v_{11} + x_2 \cdot v_{12}$, 0

fourth clock cycle: $x_1 \cdot v_{41}$, $x_1 \cdot v_{31} + x_2 \cdot v_{32}$, $x_1 \cdot v_{21} + x_2 \cdot v_{22}$, $x_1 \cdot v_{11} + x_2 \cdot v_{12} = y_1$

[2]The net dimensions have been chosen in order to clarify the figures

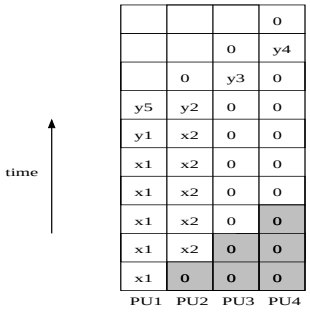

Figure 9: Data

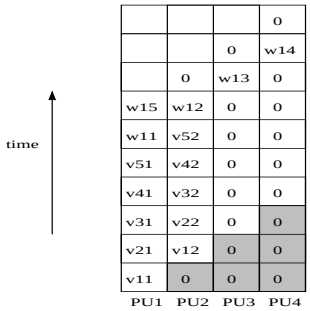

Figure 10: Weights

and, from now on, the pipeline would generate one output per clock cycle.

6 Implementation details

The system has been synthesized using a 0.35 μm technology library of standard cells provided by Mietec [1]. An active area of approximately 11.1 mm^2 without including any connection has been obtained. Due to the combinational delays, the maximum operating frequency is 111 MHz (limited by the combinational delay of the activation function generation module). For a *(m:n:p)* topology configuration, where m, n and p are multiples of the number of processing units[3], the average clock cycle count values per data have been measured and appear in table 1 (where $\pi(x) = ceil\left(\frac{n-1}{8} + 1\right)$ and $ceil(x)$ is a function rounding the value of x towards infinity).

Figures 11 and 12 depict the obtained results about learning speed (GCUPS[4] units) and recall speed (GCPS[5] units). In these figures it is shown a comparative with other digital, analog and mixed (PC-accelerators) solutions [7]. It can be seen that our system improves the speed performance of most other solutions. Only the ASICs Neuroclassifier (which

[3]This is the optimal case, for which a simple expression can be obtained

[4]Giga-Connections Updated Per Second (learning phase)

[5]Giga-Connections Per Second (recall phase)

lacks on-chip learning) and the RN-200 (with only 40 MCUPS) offer better response during the recall phase [7]. For these chips, however, there was no available data about their area dimensions and the type of arithmetic (fixed or floating point) employed.

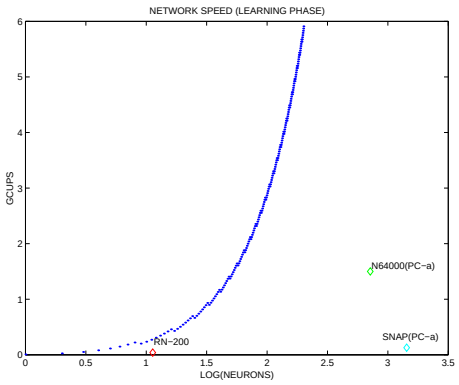

Figure 11: Chip speed during learning phase

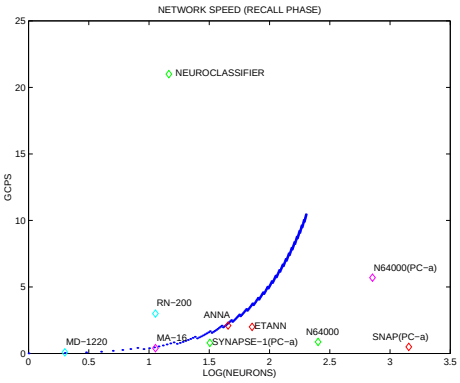

Figure 12: Chip speed during recall phase

7 *Conclusions and future work*

The design of a new hardware architecture that satisfies the Neural Network functionality with real time restrictions has been shown. This solution exhibits a high configurability and uses an efficient pipeline in order to improve the throughput of the system. However, the proposed architecture does not relinquish to the precision and flexibility of other solutions thanks to the use of floating point arithmetic. By using 24 bits of precision in coding data, a sufficient accuracy on the learning phase is achieved and this stage can be implemented on chip.

Besides, the internal memories could be written from the output of the chip. So that, a great flexibility on changing the initial weights and doing the learning off-chip could be obtained.

Additionally, thanks to a novel implementation of the sigmoid activation function (through the modification of the A-Law, based on a completely combinational module), the system does not present neither the area constraints of traditional solutions using internal LUTs nor the time delays involved in external memory accesses.

Finally, the chip features are suitable for hard constrained real-time applications like artificial vision, speech recognition and control, allowing a fast response in the learning and recall phase.

As future work, reduction of memory and bit requirements by means of novel learning algorithms like *Reactive Tabu Search* [2] (which seems to be able of working with only 2 or 3 bits coded data) should be explored. Also, the hardware could be improved by implementing several learning algorithms and activation functions that could be chosen in order to experiment with different features of Neural Nets.

References

[1] ALCATEL. *Standard cell design data book 0.35 µm CMOS*, 1998.

[2] R. Battiti. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6:1185–1200, May 1995.

[3] N. M. Botros. Hardware implementation of an artificial neural network using Field Programmable Gate Arrays (FPGA's). *IEEE Transactions on Industrial Electronics*, 41(6):665–667, December 1994.

[4] L. Fausett. *Fundamentals of neural networks. Architectures, algorithms and implementations.* Prentice Hall, 1994.

[5] R. Gadea. Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation. In *ISSS*. IEEE, September 2000.

[6] P. Ienne. Quantitative comparison of architectures for digital neuro-computers. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1987–1990, October 1993.

[7] C. S. Lindsey. Review of hardware neural networks: a user's perspective. Technical report, Physics Dept. - Frescati, Royal Institute of Technology Frescativägen 24, 104 05 Stockholm, Sweden, 1999.

[8] M. Murakawa. The GRD chip: reconfiguration of DSPs for neural network processing. *IEEE Transactions on Computers*, 48:628–639, June 1999.

[9] University of Stuttgart. *SNNS Stuttgart Neural Network Simulator. User manual, version 4.2*, 1998.

[10] S. Satyanarayana. *Analog VLSI implementation of reconfigurable neural networks.* PhD thesis, Columbia University, New York, 1991.

[11] S. Satyanarayana. A reconfigurable VLSI neural network. *IEEE Journal of Solid-State Circuits*, 27(1):67–81, January 1992.

[12] B. Sklar. *Digital communications.* Prentice Hall, 1988.

[13] N. Sundararajan. *Parallel architectures for artificial neural networks.* IEEE Computer Society, 1998.

[14] D. R. Tveter. Backpropagator's review. http://www.dontveter.com/bpr/bpr.html, March 2000.