A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function

Pedro Ferreira[1], Pedro Ribeiro[1], Ana Antunes[1], Fernando Morgado Dias[2]

[1]Escola Superior de Tecnologia de Setúbal do Instituto Politécnico de Setúbal,
Departamento de Engenharia Electrotécnica, Campus do IPS, Estefanilha, 2914-508 Setúbal, Portugal
Tel: +351 265 790000, Fax: +351 265 721869
[2]Departamento de Matemática e Engenharias, Universidade da Madeira
Campus da Penteada, 9000-390 Funchal
Tel: +351 291-705150/1, Fax: +351 291-705199
Email: pedromdsf@netcabo.pt ; {pr-apinf, aantunes}@est.ips.pt ; morgado@uma.pt

Abstract. Several implementations of Feedforward Neural Networks have been reported in scientific papers. These implementations do not allow the direct use of off-line trained networks. Usually the problem is the lower precision (compared to the software used for training) or modifications in the activation function. In the present work a hardware solution called Artificial Neural Network Processor, using a FPGA, fits the requirements for a direct implementation of Feedforward Neural Networks, because of the high precision and accurate activation function that were obtained. The resulting hardware solution is tested with data from a real system to confirm that it can correctly implement the models prepared off-line with MATLAB.

Keywords: FPGA, Feedforward Neural Networks, Hardware, VHDL.

1. INTRODUCTION

Artificial Neural Networks (ANN) became a common solution for a wide variety of problems in many fields, such as control and pattern recognition to name but a few. Within these solutions a large share were developed using one type of ANN that only allow connections in the forward sense (that is, between one layer and the next one closer to the output), called Feedforward Neural Networks (FNN). It is therefore not surprising that some of the solutions have reached an implementation stage where specific hardware is considered to be a better solution than the most common implementation within a personal computer (PC) or workstation.

A number of reasons can be pointed out as the motivation for this:

- Need for higher processing speed.
- Reduced cost for each implementation.
- Reliability.

The PC or the workstation being conventional von-Neumann architectures are not able to provide the high processing speed required by many applications. A hardware solution can also be less expensive and more reliable than a PC.

Also as pointed out by (Lippmann, 1987) "The greatest potential of neural networks remains in the high-speed processing that could be provided through massively parallel VLSI implementations".

Within the possible solutions: analog, digital or hybrid, as it is possible to find among the commercial implementations (Dias et al. 2003), the digital one has the following advantages:

- Weight storage in memory.
- Facility of integration with other applications.
- Facility to implement learning algorithms.
- Exact within the number of bits of the operands and accumulators.
- Low sensivity to electric noise and temperature.

Considering the possible solutions for a digital implementation there are still choices to make: full custom ASICs, Sea of Gates, FPGAs (to name only a few that have already been used in ANN implementation). The real choice is in fact quite reduced since for most of the applications a specific hardware solution must be designed and therefore it is impossible to use the same design for many applications, which makes it economically unacceptable to use ASICs or Sea of Gates solutions.

This leads, in the case of a specific application, to a FPGA solution because the cost associated with only a few copies of the hardware is acceptable. In the literature it is possible to verify that several solutions have already been tested in the FPGA context (Lysaght et al. 1994), (Arroyo Leon et al. 1999), (Skrbek, 1999), (Wolf et al. 2001), (Ayala et al. 2002), (Bade et al. 1994), (Economou et al. 1994).

Nevertheless, the solutions that were found do not allow the direct use of the neural models that are prepared frequently with software (like MATLAB or specific software for ANN) within PCs or workstations.

All the solutions that the authors were able to verify within FNN present either a much lower precision when compared with these software solutions (Lysaght et al. 1994), (Arroyo Leon et al. 1999), (Economou et al. 1994) or modifications in the activation function that make them unacceptable to use directly the weights previously prepared (Skrbek, 1999), (Wolf et al. 2001), (Economou et al. 1994).

Filling this gap between software and hardware solutions, allowing the direct use of the weights is an important step in the development of the ANN field.

In the present work a hardware solution called Artificial Neural Network Processor (ANNP), using a FPGA, developed to fit the above requirements for a specific application is presented.

Although the hardware was developed with a specific application in view, the following characteristics can be pointed out:

- Scalability, since the ANNP can be used for different network sizes.
- High precision, since the inputs and internal calculations are done with 32 bits in floating point notation.
- Accurate activation function, since the highest error allowed in the implemented activation function is of 2.18×10^{-5}.

2. HARDWARE IMPLEMENTATION

There are three main problems that must be addressed when a hardware implementation with a FPGA is considered:

- Notation
- Activation function
- Device capabilities

Notation is a key issue in the hardware implementation. It is almost consensual that floating point notation should be used if a high precision is seeked with a lower number of bits. The problem here is the complexity resulting from the necessary operations (multiplication, division, addition) with this notation.

The activation function is particularly difficult to implement in the case of the sigmoid functions. The direct implementation, for example, of the hyperbolic tangent would require an adder, multiplier and exponential.

The problems pointed out can be solved if enough hardware is available, this is why the device capabilities limit the possible solutions.

Notation

The notation chosen was 32 bits floating point according to the IEEE 754-1985 standard. Although it has been stated in (Zhu et al. 2003) that "A few attempts have been made to implement ANNs in FPGA hardware with floating point weights. However, no successful implementation has been reported to date." and (Nichols et al. 2002) have concluded that "floating point precision is still not feasible in FPGA based ANNs", there were at least two applications reported: (Arroyo Leon et al. 1999) which used floating point notation of 17 bits and (Ayala et al. 2002), which used 24 bits.

The fixed-point notation would require a larger number of bits to obtain the same precision and maximum number to be represented and for this reason this option was discarded.

Other solutions include pulse stream arithmetic (Lysaght et al. 1994) or bitstream (Bade et al. 1994).

Activation Function

In the present application, the activation function used was the hyperbolic tangent (Equation 1).

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (1)$$

This equation can be re-arranged so that it has only one exponential but it still requires the operations of adding, dividing and the calculation of the exponential itself.

Understanding that the direct implementation is not suitable for the present solution, the hyperbolic tangent was studied in order to simplify its implementation.

As the objective of the present work was defined from the start, a maximum error could be set and a classical approach with a Look-Up-Table (LUT) was tested, but it was easily verified that this solution would be too expensive in hardware.

A new approach was then tested: piece-wise linear approximation. This corresponds to one of the classical solutions for the implementation of the activation function (Zhu et al. 2003), but in this case with an important variation. While most of these solutions are done using only three linear sections to approximate the hyperbolic tangent, the decision here was to use as many sections as necessary to implement it with the previously defined precision.

With this objective a study was carried out to prepare the linear sections and determine how many were needed. After reducing the hyperbolic tangent to the small part that needs to be represented, this was done in the following way:

1. Choose the first linear section (from initial point of the region to be represented choose the best fit linear section).
2. Set a maximum allowed error.
3. Compare this linear approximation with the hyperbolic tangent implemented in MATLAB in order to verify when the maximum allowed error is met.
4. From the previous point, start a new linear section.
5. Repeat the operation until the region needed was fully represented (the function is even so only half of the representation is needed).

This algorithm led to the representation of the hyperbolic tangent with 256 linear sections (only the linear section parameters are stored) and provides a maximum error of 2.18×10^{-5} in values that are in the range of [-1,1] (function output). The number of 256 sections was obtained choosing the smaller power of 2 (see below that the binary search algorithm is used) that allowed meeting the specification of maximum allowed error defined.

This can be compared with other solutions like the one using the Taylor series used in (Arroyo Leon et al. 1999), which obtained an error of 0.51% and piecewise-linear approximation used in (Ayala et al. 2002), which obtained 0.0254 of "standard deviation with respect to the analytic form".

It is worth to verify that to obtain this error with the classical LUT approach 18110 samples of the function were needed. These samples, represented in the 32 bits notation used, would require more than a single FPGA of the type used, just to represent the activation function.

The binary search algorithm was used for searching the parameters of the linear section to be used. This algorithm allows finding the parameters in a maximum of eight clock cycles by splitting the intervals in half

and minimizes the time differences for different searches.

Figure 1 shows a detail of the linear section approach versus the single point approach.

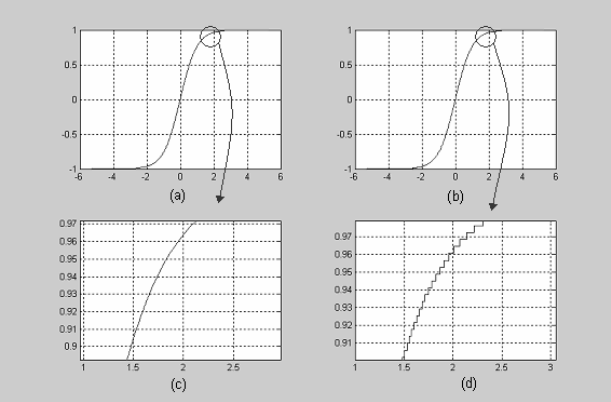

Figure 1 – Hyperbolic tangent: (a) linear section approach; (b) single point approach; (c) detail of the linear section approach; (d) detail of the single point approach.

Hardware Platform

The hardware platform used is the Cyclone EP1C20F324C7 FPGA from ALTERA, in the kit Cyclone SmartPack from Parallax that can be seen in Figure 2. This board was chosen because it was the less expensive one available that could hold the implementation.

This FPGA is composed of 20060 Logic Elements (LE), 294912 bits of memory and 2 Phase Locked Loops. The Cyclone SmartPack kit includes a RS-232 port, Joint Test Action Group (JTAG) port, 8 Mbit of flash memory to store complete device configuration and memory contents for power-up boot, a 50 MHz oscillator and 128 Input/Output pins.

The development of the circuit was done in the Quartus II environment.

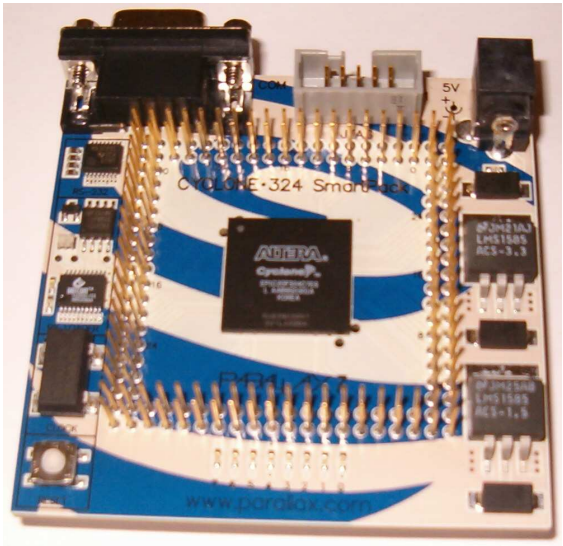

Figure 2 – Picture of the Cyclone SmartPack kit.

Implementation

The device capabilities limit the choices that can be made since in the end the design chosen must fit in the hardware.

The high precision chosen limits the amount of hardware that is possible to fit in the FPGA and the decision was to reduce the hardware to the minimal that is needed to implement the FNN.

The ANNP was developed using the VHDL language and implements the following components:

- 32 bits multiplier in floating point notation.
- 32 bits adder in floating point notation.
- Activation function.
- Memory blocks.
- Three finite state machines.
- Serial communication using RS-232 protocol.

The memory blocks are used to hold the weights, the inputs and partial and final results. The three finite state machines are used to control the operation of the ANNP and input and output data through the RS-232 port.

The basic structure is composed of the multiplier, adder and activation function (Figure 3) and must be reused several times in order to implement the ANN.

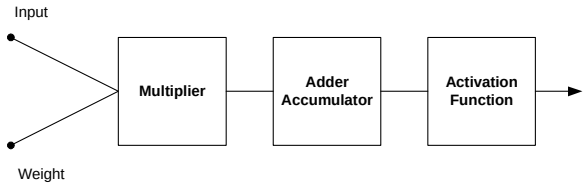

Figure 3 – Logical structure of the basic processing block in the ANNP.

The ANNP is prepared to implement ANNs up to 16 neurons per layer and up to 17 layers.

The number of LEs used is 4446, which means 22% of the total number of LEs available.

The implementation of the activation function uses three ROMs of 256x32 bits, that is 24576 bits, which are implemented in 6 blocks of 256x16 bits.

The complete implementation uses 235008 bits, which are in fact 285696 bits if the parity bits (and others that are used for special functions) are taken into account, of a total amount of 294912 bits available, that means that 96.9% of the bits were used (only 2 blocks of 4096 bits were left unused) and this was the real limitation to implement a larger architecture that could have been expanded in the number of basic processing blocks used or by enlarging the functional capacity of this block.

Within the total memory used most of it is for storing the weights since the ANNP has reserved memory for a maximum amount of 6416 weights, including the ones for the bias.

The maximum number of inputs that the ANNP is prepared to process is 16.

The serial communication protocol, RS-232 is set to 115200bps, since the communication is done with one start and one stop bit, it means 11520Bps.

3. TEST SYSTEM

The hardware implementation of the ANN was tested using several models of a system, which were previously prepared in MATLAB.

This system is a reduced scale prototype kiln, which was working under measurement noise. For further details please see (Dias et al. 2001).

To enhance debugging during the test phase a software application was developed, which is capable of controlling the operation of the ANNP step-by-step and giving information about the internal state of the variables and registers.

The main window of this application that runs in MATLAB can be seen in Figure 4.

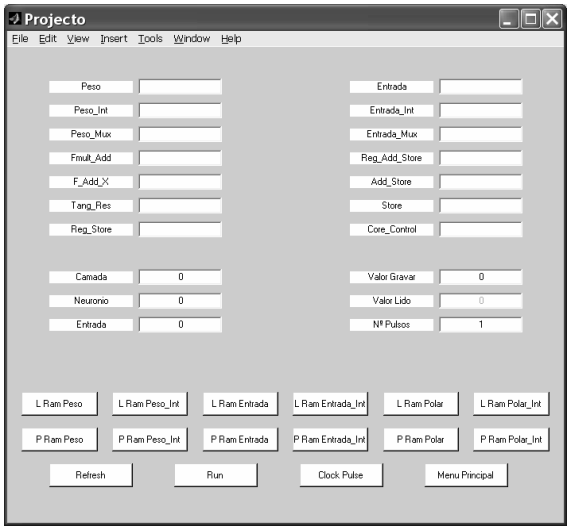

Figure 4 – Main window of the debugging application.

This application is also used to send data and commands to the FPGA and to receive the results. The communication between MATLAB and the FPGA is done using frames of 48 bits that send commands and data, while the communication between the FPGA and MATLAB is done using frames of 32 bits that contain the response to the previous command sent.

With the communication rates implemented for the RS-232 standard, this results in a speed of 1920 weights per second, this means that a network of, for example 100 weights, will take 0.052 seconds to be configured.

4. RESULTS

Three sets of models were used in testing the hardware. The models consist of couples (one direct and one inverse model) with different number of inputs and neurons in the hidden layer. All the models have only one hidden layer with hyperbolic tangent as activation function and linear activation function in the output layer.

The method chosen to verify the quality of the solution developed was to implement the inverse models of the system in the FPGA and the direct model of the system in MATLAB, this way the model in MATLAB would act as the system and it would be possible to perform control action. The control strategy used was a simple Direct Inverse Control (DIC) that is represented in Figure 5.

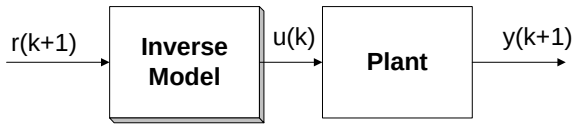

Figure 5 – Block diagram for Direct Inverse Control.

Figure 6 shows a block diagram of the connection between MATLAB and the FPGA and Table 1 shows the characteristics of the inverse models tested in the FPGA.

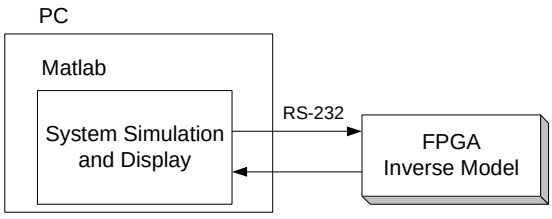

Figure 6 – MATLAB and FPGA connection.

In Figures 7, 8 and 9 the control results of DIC with three different references can be seen. In Figure 7 a standard ramp is used as set point, in Figure 8 the reference is a ramp with a square wave signal and in Figure 9 a pseudo-random signal is used. These are results for the set of models referred as Model 1.

To complete the analysis of the results and separate the error that is introduced by the use of the FPGA from the one introduced by the models, the same control is performed with both models in MATLAB.

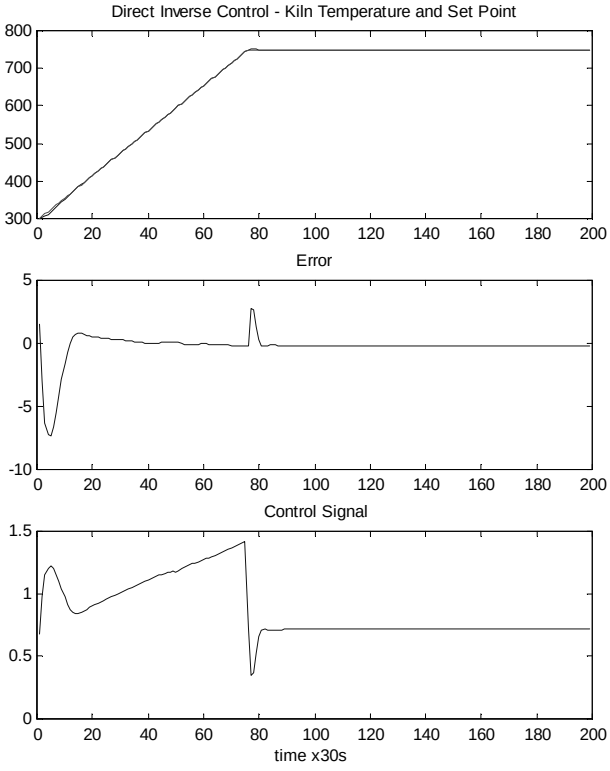

Figure 7 – FPGA results with ramp signal.

The results in the form of Mean Square Error (MSE) are summarized in Table 2. As can be seen the difference obtained between the control with the FPGA and the ones simulated in MATLAB, in terms of MSE, does not exceed $5x10^{-8}$.

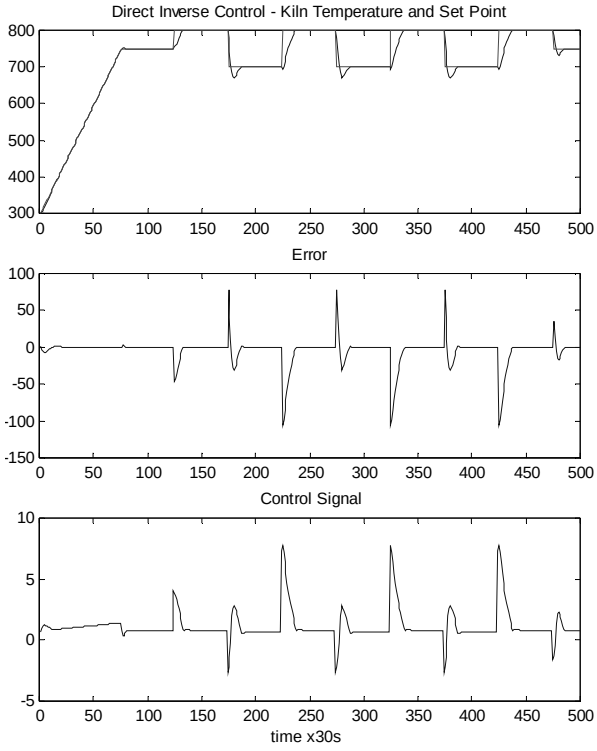

Figure 8 – FPGA results with ramp and square wave signal.

Table 1- Resume of the characteristics of the inverse models tested in the FPGA.

Model Name	Inputs	Neurons
Model 1	4	6
Model 2	2	3
Model 3	4	4

5. CONCLUSION and FUTURE WORK

This work proposes a hardware implementation of an ANN using a FPGA. The FPGA was chosen because of the lower prices for a single implementation.

The goal that was searched was to obtain a hardware solution that allowed the direct use of the weights, usually prepared in a software environment with a much higher precision than the ones usually obtained in FPGAs implementation.

This objective was successfully accomplished as was shown by the control tests done with models of a real system, where the maximum error obtained between the MATLAB and the hardware solution, using the MSE as a measure was of $5x10^{-8}$.

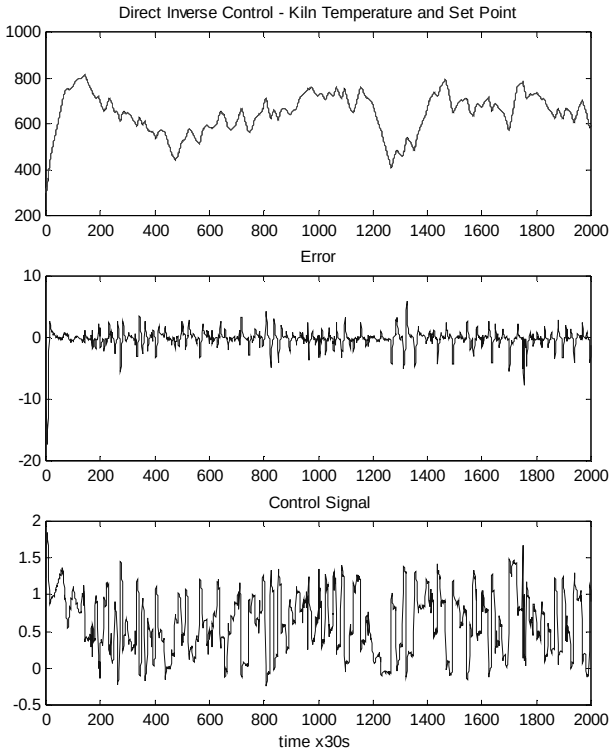

Figure 9 – FPGA results with pseudo-random signal.

A new algorithm to apply with piece-wise linear approximation was also presented that allowed high precision in the implementation of the hyperbolic tangent. This algorithm allows the definition of the maximum error that is acceptable and supplies the number and parameters of the equations of the corresponding linear sections.

As further work the authors would like to maintain the objective that led to this implementation, that is high precision, and develop solutions that enable the full processing of a neuron or a full layer at once. With the further increase in the FPGAs density this seems a real objective in short term.

Table 2- Mean Square Error comparison between the MATLAB and FPGA results.

Model Name	MATLAB Results			FPGA Results		
	Ramp	Ramp and square wave	Pseudo-Random	Ramp	Ramp and square wave	Pseudo-Random
Model 1	$1.2146x10^{-4}$	0.0146	$1.3343x10^{-4}$	$1.2151x10^{-4}$	0.0146	$1.3345x10^{-4}$
Model 2	$7.8682x10^{-5}$	0.0040	$1.3445x10^{-4}$	$7.8666x10^{-5}$	0.0040	$1.3440x10^{-4}$
Model 3	$8.1468x10^{-6}$	0.0069	$1.5426x10^{-5}$	$8.1456x10^{-6}$	0.0069	$1.5426x10^{-5}$

6. ACKNOWLEDGMENT

The authors would like to acknowledge the ALTERA University program for support and software used in this work.

7. REFERENCES

Lippmann, R. P., "An introduction to computing with neural nets", IEEE ASSP Magazine, 4-22, 1987.

Dias, F. M., A. Antunes and A. M. Mota, "Commercial Hardware for Artificial Neural Networks: a Survey", SICICA - 5th IFAC International Symposium on Inteligent Components and Instruments for Control Applications, Aveiro, 2003.

Lysaght, P., J. Stockwood, J. Law and D. Girma, "Artificial Neural Network Implementation on a Fine-Grained FPGA", in R. W. Hartenstein, M. Z. Servit (Eds.) Field-Programmable Logic, Springer Verlag, pp. 421-431, Czech Republic, 1994.

Bade, S. L. and B. L. Hutchings, "FPGA-Based Stochastic Neural Networks – Implementation", IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, pg.189-198, 1994.

Zhu, J. H. and Peter Sutton, "FPGA implementations of neural networks – a survey of a decade of progress", 13th International Conference on Field Programmable Logic and Applications, Lisbon, 2003.

Arroyo Leon, M. A., A. Ruiz Castro and R.R. Leal Ascenccio, "An artificial neural network on a field programmable gate array as a virtual sensor", Proceedings of The Third International Workshop on Design of Mixed-Mode Integrated Circuits and Applications, Puerto Vallarta, Mexico, pp. 114-117, 1999.

Ayala, J. L., A. G. Lomeña, M. López-Vallejo and A. Fernández, "Design of a Pipelined Hardware Architecture for Real-Time Neural Network Computations", IEEE Midwest Symposium on Circuits and Systems, USA, 2002.

Nichols, K., M. Moussa and S. Areibi, "Feasibility of Floating-Point Arithmetic in FPGA based Artificial Neural Networks", CAINE, San Diego California, pp:8-13, 2002.

Skrbek, M., "Fast neural network implementation", Neural Network World, vol.9, nº5, pp.375–391, September 1999.

Wolf, D. F., R. A. F. Romero and E. Marques, "Using Embedded Processors in Hardware Models of Artificial Neural Networks", V Simposio Brasileiro de automação inteligente, Brasil 2001

Dias, F. M. and A. M. Mota, "Comparison between different Control Strategies using Neural Networks", 9th Mediterranean Conference on Control and Automation, Dubrovnik, Croatia, 2001.

Economou, G. - P. K., E. P. Mariatos, N. M. Economopoulos, D. Lymberopoulos, and C. E. Goutis "FPGA Implementation of Artificial Neural Networks: An Application on Medical Expert Systems", Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, pp:287 - 293, 1994.

Pedro Ferreira and Pedro Ribeiro are former students of the Superior School of Technology of Setubal. They hold a five year engineering diploma in electronics and Computers.

Ana Antunes received her *Diplôme D'Études Approfondies* in Microelectronics from the University Joseph Fourier in Grenoble, France (1995) and currently teaches at the Superior School of Technology of Setubal, Portugal and is preparing her PhD degree at the University of Aveiro,

Fernando Morgado Dias received his *Diplôme D'Études Approfondies* in Microelectronics from the University Joseph Fourier in Grenoble, France (1995) and his PhD from the University of Aveiro, Portugal and currently is a Assistant professor at the University of Madeira.