A New Solution for the Hyperbolic Tangent Implementation in Hardware: Polynomial Modeling of the Fractional Exponential Part

Ivo Nascimento, Ricardo Jardim and Fernando Morgado-Dias

Competence Center of Exact Science and Engineering and Research Center of Mathematical Sciences, University of Madeira, 9000-390 Funchal, Portugal

+351 291705307; +351 291705199;

akronic@gmail.com, rjjardim@gmail.com and morgado@uma.pt

Abstract: The most difficult part of an Artificial Neural Network to implement in hardware is the non-linear activation function. For most implementations, the function used is the hyperbolic tangent. This function has received much attention with relation to hardware implementation. Nevertheless, there is no consensus regarding the best solution. In this paper we proposes a new approach by implementing the hyperbolic tangent in hardware with a polynomial modeling of the fractional exponential part. The results in the paper then demonstrates, through the use of an example, that this solution is faster than the CORDIC algorithm, but slower than the piecewise linear solution with the same error. The advantage over the piecewise linear approach is that it uses less memory.

Keywords— Algorithms implemented in hardware, Gate arrays, Hardware description languages, Neural networks, CORDIC.

1 Introduction

Artificial Neural Networks (ANN) are biologically inspired models that compute a simple mathematical function aimed at mimicking the capacities of living beings.

These simplified models have found a wide range of applications, covering areas ranging from engineering to economy, where they are applied for control, modeling and prediction in the most diverse situations.

In the first stage, ANN are implemented in a Personal Computer (PC). However, for some industrial applications this is not convenient and dedicated hardware is required. The need for physical implementations also arises in medical applications of ANN which work inside or in close connection with the body. The work referred in [14] contains many examples of neural vision systems that

cannot be used attached to computers. Other examples of physical implementation of ANN can be found in [15], where an ANN hardware solution for Bladder Cancer detection is proposed; [16-17] refer the for the H1 electon-proton collider experiment controlled by ANN; [18] which describes Falcon, a detection system for payment card fraud.

For a broader overview of more than two decades of hardware implementations of ANN and their use please refer to [19-20].

This hardware can range from a microcontroller to an Application Specific Integrated Circuit (ASIC). Both have disadvantages, the former usually being too slow and the latter too expensive. The intermediate solution, in order to perform the calculations needed for ANN, and thus achieve a considerable processing speed and an acceptable price, is obtained by Field Programmable Gate Arrays (FPGA). This is the option utilized in the largest share of hardware implementations of ANN and will also be used in this work.

The hardware implementation of ANN faces some difficult issues: a large number of calculations, a high number of bits and most importantly the implementation of the nonlinear activation function used in most of the ANN.

For the last of these problems this paper proposes a new alternative: a Polynomial Modeling of the Fractional Exponential Part (PMFEP), in order to implement the hyperbolic tangent. This solution, as will be shown further on, achieves equilibrium between speed and the amount of memory to be used. Moreover, it allows for a high degree of accuracy and has an additional advantage of having less break points than the piecewise linear approaches, thus making it smoother.

2 State of the art

Many different solutions have been tested for implementing the activation functions of ANN in hardware of which most of the effort invested has been placed in implementing the hyperbolic tangent, due to the use of this function by a large share of the implementations.

The implementations found in the literature usually use oversimplified activation functions and their results are reported in a vague form and are difficult to reproduce. Nevertheless, the best solution can be found in [1-2], where a maximum error of $2{,}18 \times 10^{-5}$ and a mean square error of 5×10^{-8} in a control simulation was obtained, using linear sections in which the sections were selected

and distributed (though not evenly) to meet a maximum pre-defined error. In [3], the solution is based on a Taylor series which achieved an error of 0,51%; in [4], a piecewise linear implementation is proposed which obtained 0.0254 of “standard deviation with respect to the analytic form”; and in [5], a set of seven polynomial 5th order approximations is proposed for a maximum error of $7x10^{-5}$.

The CORDIC algorithm has also been tested in [6-8, 12] as an alternative solution for the hyperbolic tangent. The direct use of this algorithm for calculating the hyperbolic functions does not guarantee convergence, but it can be used with the proper restriction of the angles to be applied. Nevertheless the calculation of the hyperbolic tangent is obtained from the hyperbolic sin and cosine functions [13].

There are also other proposals such as a fuzzy determined function, as in [9-10], which was inspired by an earlier proposal [11]. The tests developed by the research team indicate that this proposal, nevertheless, presents a high maximum error, greater than 0.4.

The impact of error introduced by the hardware implementation depends on the task where the ANN is being used. For classification tasks error increase might be unacceptable although small changes in the ANN might not reflect in the output. For regression tasks the error introduced should be, at most, of the same order than the noise affecting the data used for training the ANN. Otherwise the models used will deviate from the behavior of the system they represent and might result in severe problems. As an example, if these models are used in control loops, the loop might fail to stabilize.

3 Hyperbolic Tangent Replacement and Proposed Solution

Our goal with this work is to obtain a hardware implementation of the hyperbolic tangent that does not introduce a large error when compared to the software counterpart. This is necessary for most applications and as can be seen from the previous section the solutions proposed until now are not precise enough and the results are presented in such an imprecise way that they cannot even be compared.

The additional requirements are that the hardware necessary for such an implementation is not excessively large and the price for such an implementation is acceptable.

In order to replace the hyperbolic tangent several tests were performed: the second order polynomial approximation proposed in [10] that can be seen in equation (1), up to forth order polynomial approximation of the hyperbolic tangent, up to fifth order approximation of the exponential function and piecewise linear according to [1-2]. The polynomial approximations developed by the research team were based in the least squares algorithm.

$$f(x) = \begin{cases} sign(x) & if\ |x| > L \\ \dfrac{-x.|x|}{L^2} + \dfrac{2.x}{L} & otherwise \end{cases} \qquad (1)$$

The exponential approximation was abandoned because the results were very poor and the piecewise linear solution can be tailored to any given error. For the remaining solutions a summary of the results obtained in terms of maximum, minimum and average error (using a 10000 point test set) can be seen in table 1.

TABLE I Error obtained for the initial set of solutions

Solution	Error		
	Max.	Min.	MSE
Equation 2 (L=2)	0.0432	-0.0432	$2.7475*10^{-4}$
4^{th} order polynomial	0.0188	0.0098	$5.1396*10^{-5}$

After verifying that the previous tests were not satisfying, a new approach was tested and the results are presented in the next sub-section.

A. Polynomial Modeling of the Fractional Exponential Part

The proposal presented in this paper took form observing that the exponential function could be decomposed into an integer and a fractional part:

$$e^{x} = e^{x\text{int}} \times e^{xfrac} \qquad (2)$$

Where xint represents the integer part of x and xfrac represents the fractional part of x. The fractional part has a repetitive behavior and can be modeled on its own, while the integer part can be pre-calculated and stored in a Look Up Table (LUT), thus avoiding additional calculation.

Since the behavior of the fractional part is not very nonlinear, it can be represented easily by a polynomial equation without a large error.

The research team tested polynomials of different degrees, with the resulting values presented in table II.

TABLE II -Error obtained for the different orders tested for the PMFEP

Polynomial order	Error
3	$4.700284*10^{-4}$
4	$2.498336*10^{-5}$
5	$9.989812*10^{-7}$
6	$5.690028*10^{-8}$
7	$2.689197*10^{-8}$

The solution chosen is a fifth degree polynomial and can be seen in figure 1 (for this error it is not possible to see the difference between the polynomial and the real value). Augmenting the polynomial order would result in a smaller error but would be more time-consuming while calculating each value. Thus, the final

solution is a compromise between precision and the delay introduced by the calculation.

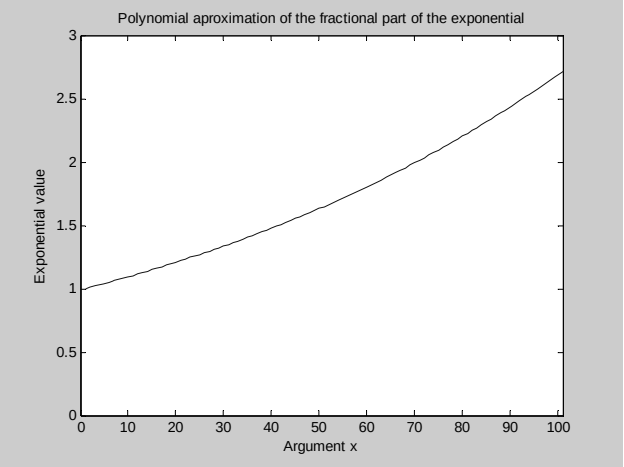

Fig. 1. Representation of the fractional part of the exponential function.

To implement this solution in hardware, it is necessary to first split the integer and fractional part of the argument, then calculate the fractional part and retrieve the integer part of the exponential from the LUT, calculate the full value of the exponential and finally, by using this value, obtain the hyperbolic tangent value. The fluxogram for the calculation of the hyperbolic tangent using this principle is shown in figure 2.

The implementation was developed in Verilog and makes use of the odd function property and saturation while using 14 values in the LUT (e^0 to e^{13}).

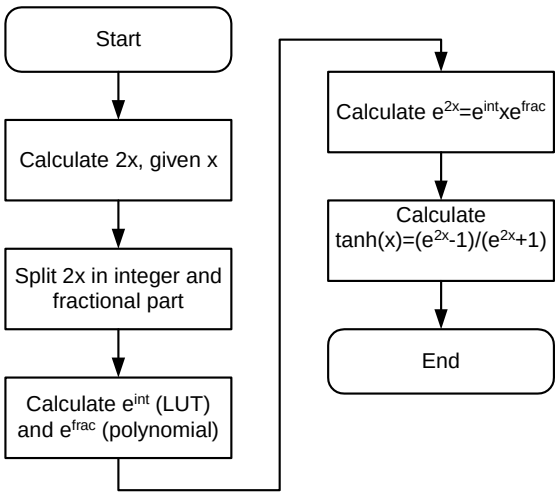

Fig. 2. Block diagram of the algorithm used for calculating the exponential in the proposed solution.

4 Results Obtained

A new proposal for the hardware implementation of the hyperbolic tangent, such as the one presented here, must be proven to be efficient, in order to be worth considering. In order to do so, the error needs be analyzed and the resources used need to be calculated and compared against other alternatives.

The implementation of the hyperbolic tangent, using the fifth order polynomial for the approximation of the fractional part, obtained a maximum error of 9.989812×10^{-7} with 1000 points, while using internal calculations with 32 bits. It took 40 clock cycles to calculate the output value.

In this paper, a XC5VFX130T FPGA was used. Table III shows a summary of the resources needed for this solution, only for the hyperbolic tangent. These results were obtained mapping the 5^{th} order PMFEP description in Very high speed integrated circuit Hardware Description Language (VHDL) to the FPGA with the Integrated Software Environment from Xilinx.

TABLE III - Resources used in the PMFEP implementation

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	414	81,920	1%
Number used as Flip Flops	413		
Number used as Latch-thrus	1		
Number of Slice LUTs	3,429	81,920	4%
Number used as logic	3,420	81,920	4%
Number using O6 output only	3,387		
Number using O5 output only	2		
Number using O5 and O6	31		
Number used as exclusive route-thru	9		
Number of route-thrus	29		
Number using O6 output only	10		
Number using O5 output only	18		
Number using O5 and O6	1		
Number of occupied Slices	949	20,480	4%
Number of LUT Flip Flop pairs used	3,485		
Number with an unused Flip Flop	3,071	3,485	88%
Number with an unused LUT	56	3,485	1%
Number of fully used LUT-FF pairs	358	3,485	10%
Number of unique control sets	10		
Number of slice register sites lost to control set restrictions	19	81,920	1%
Number of bonded IOBs	66	840	7%
IOB Flip Flops	32		

Number of BUFG/BUFGCTRLs	2	32	6%
Number used as BUFGs	2		
Number of DSP48Es	2	320	1%
Average Fanout of Non-Clock Nets	5.12		

5 Comparison with other Solutions

Each solution present in the literature has desirable and undesirable aspects.

The piecewise linear approximation, while quick in execution, is not smooth [12], showing many points where the left and right derivatives are different.

In the CORDIC algorithm each additional stage in the calculation increases the accuracy, but also increases the time required to complete the calculation by one or more clock cycles [12]

A. Piecewise linear

The algorithm proposed in [1] can be adjusted to the desired error, by selecting a new linear section when the error meets the desired limit. As a result, the sections will be unevenly distributed, but fewer sections are required. This algorithm stores linear sections of the format y=mx+b, by using one LUT for m and b values, and using the binary search algorithm to find the section corresponding to the input value.

Using this solution, the research team prepared an implementation that has the same maximum error as the new proposal at 1000 samples.

Table IV shows a comparison of the errors between PMFEP and the piecewise algorithm used with 10000 samples. Please notice that the initial error, similar in both solutions, was measured with 1000 samples and for a deeper analysis 10 times more samples are used.

TABLE IV - Error obtained for the different orders tested for the PMFEP

Solution	Error		
	Largest Positive	Largest Negative	MSE
Piecewise Linear Algorithm	0.0432	-0.0432	$2.7475*10^{-4}$
PMFEP	0.0188	-0.0098	$5.1396*10^{-5}$

The summary of the resources used by the piecewise linear approach under these conditions can be seen in table V.

TABLE V - Resources used in the piecewise linear implementation

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	84	81,920	1%
Number used as Flip Flops	83		
Number used as Latch-thrus	1		
Number of Slice LUTs	710	81,920	1%
Number used as logic	710	81,920	1%
Number using O6 output only	691		
Number using O5 and O6	19		
Number of route-thrus	1		
Number using O5 output only	1		
Number of occupied Slices	331	20,480	1%
Number of LUT Flip Flop pairs used	745		
Number with an unused Flip Flop	661	745	88%
Number with an unused LUT	35	745	4%
Number of fully used LUT-FF pairs	49	745	6%
Number of unique control sets	6		
Number of slice register sites lost to control set restrictions	13	81,920	1%
Number of bonded IOBs	67	840	7%
IOB Latches	32		
Number of BlockRAM/FIFO	6	298	2%
Number using BlockRAM only	6		
Number of 36k BlockRAM used	6		
Total Memory used (KB)	216	10,728	2%
Number of BUFG/BUFGCTRLs	2	32	6%
Number used as BUFGs	2		
Number of DSP48Es	2	320	1%
Average Fanout of Non-Clock Nets	4.22		

This piecewise linear implementation uses the binary search algorithm and the number of clock cycles is dependent on the number of values stored in the LUT. This particular implementation takes 24 clock cycles to calculate the hyperbolic tangent approximation.

The comparison of the resources used is not straightforward. One of the key issues for these implementations is the amount of memory used. This is probably the simplest comparison which could be made.

The PMFEP uses mostly LUTs, while the piecewise linear solution uses ROM memory to store the linear section information.

The LUTs used in the XC5VFX130T have 6 inputs. This corresponds to 2^6=64 bits of information which can be stored in each LUT.

A simple comparison of the amount of memory used can be achieved by converting the capacity of the LUT in bits and by comparing the amount of bits used in each solution.

The memory used for each solution is:

PMFEP:

Number of LUT: 3485-56=3429.

Equivalent memory: 64x3429=219456 bits.

Piecewise linear:

Number of LUT: 745-35=710.

Equivalent memory: 64x710=45440 bits.

ROM: 216KBytes

Equivalent memory: 8x216000=1728000 bits.

Total memory used: 1773440 bits.

Through this simple calculation one can conclude that the piecewise linear solution uses 8 times more memory than the PMFEP solution.

B. Cordic

The original CORDIC algorithm was developed to make the conversion of rectangular coordinates into polar ones and the inverse transformation, without multipliers using shift and add operations.

Given an initial vector (x,y), for which the transformation is needed and a rotation angle θ, the relation between the initial vector and the resulting one is given by:

$$x' = x\cos\theta - y\sin\theta = \cos\theta(x - y\tan\theta)$$
$$y' = y\cos\theta + x\sin\theta = \cos\theta(y + x\tan\theta) \qquad (2)$$

The rotation angle can be decomposed into a set of small angles in powers of 2. This allows the rotation to be performed in small steps, with θ calculated according to (3):

$$\theta \cong \sum_{0}^{N} \delta_i \theta_i \qquad (3)$$

where $\delta_i = \{1,-1\}$ and defines the add or subtraction operation.

The angles are chosen as:

$$\theta_i = \tan^{-1} 2^{-i} \qquad (4)$$

and the new coordinates, after each iteration, are given by (5): ~

$$x_{i+1}' = \cos\theta_i (x_i' - y_i' \delta_i 2^{-i})$$
$$y_{i+1}' = \cos\theta_i (y_i' + x_i' \delta_i 2^{-i}) \qquad (5)$$

Considering (6) and (7)

$$k_i = \cos\theta_i = \frac{1}{\sqrt{1+2^{-2i}}} \quad (6)$$

$$K = \prod_{o}^{N} k_i \quad (7)$$

where N represents the precision needed and the number of bits used in the fractional part, the calculation in each iteration is given by (8):

$$\begin{aligned} x_{i+1} &= x_i - y_i \delta_i 2^{-i} \\ y_{i+1} &= y_i + x_i \delta_i 2^{-i} \end{aligned} \quad (8)$$

K is multiplied by the final values of x and y.

Analyzing the best CORDIC implementation, it can be concluded that with IEEE-754 32 bit notation there are 24 mantissa bits. For each bit, there is at least the delay of one clock cycle in the calculation. The calculation also requires obtaining the hyperbolic sine and cosine values and a division to be performed. Even though the sine and cosine functions can be performed simultaneously, introducing more hardware the division uses the same hardware block as in the PMFEP, and takes 24 clock cycles. If a faster divider is used, it will improve both solutions in the same way. In total, at least 48 clock cycles will be used for the CORDIC algorithm, 8 more than with the PMFEP.

Figure 3 shows a graphic to compare the three solutions analyzed. The horizontal axis shows the delay for each solution, while the vertical axis compares the memory resources required. The CORDIC line is shown dashed, since the resources for this solution are estimated, while the remaining resources have been measured in the implementation.

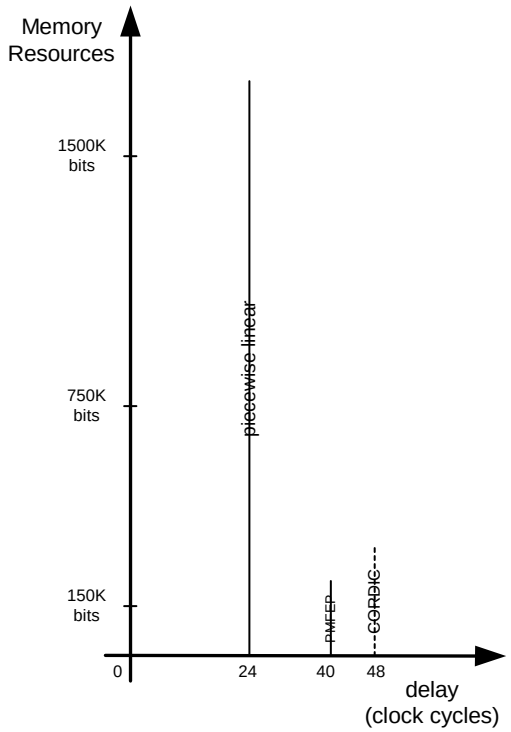

Fig. 3. Representation of the memory resources versus delay of each solution.

The product of delay times memory resources can be calculated and demonstrates, for this example, that the PMFEP solution is less demanding than the piecewise linear solution:

TABLE VI – Product Delay - memory comparison for the different solutions

Solution	Product Delay - Memory
Piecewise Linear Algorithm	41472000
PMFEP	8778240
CORDIC	superior to 10533888

The last value was estimated, considering the 48 clock cycles required to calculate the output and, at least, the same amount of memory used in PMFEP.

6 Conclusion

In the ANN world many applications have been presented. Most of these applications use hyperbolic tangents as activation functions and some have

reached a hardware implementation phase. The frequently used hyperbolic tangent creates a real problem in terms of accurate hardware implementation. This paper presented a new alternative: Polynomial Modeling of the Fractional Exponential Part.

For this new solution this paper has shown, through an example, that it uses less clock cycles than the CORDIC algorithm, but more than a piecewise linear solution: while it can use less memory than the piecewise linear alternative for the same error.

For the CORDIC and the PMFEP, it is possible to pipeline the calculations within each layer, thus reducing the delay to one complete calculation in addition to the last part of the pipeline for each additional value to be calculated (this would depend on the level of pipelining achieved).

The PMFEP solution could be understood as a solution which stands midway between the piecewise linear and the CORDIC algorithm: using less memory than the piecewise linear, and is faster than the CORDIC algorithm.

With the reduction of the resources required for the implementation and the low error introduced, PMFEP allows for the direct use of high accuracy software-trained ANN.

7 References:

[1] P. Ferreira, P. Ribeiro, A. Antunes, and F. M. Dias ,"A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function", Neurocomputing, Vol. 71, Issues 1-3, Pages 71-77, December 2007.

[2] P. Ferreira, P. Ribeiro, A. Antunes and F. M. Dias, "Artificial Neural Networks Processor - a Hardware Implementation using a FPGA", Field-Programmable Logic and its Applications, Belgium, LNCS-3203, 2004.

[3] M. Leon, A. Castro and R. 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, 1999, pp. 114–117.

[4] J.L. Ayala, A.G. Lomena, 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.

[5] A.M. Soares, J.O.P. Pinto, B.K. Bose, L.C. Leite, L.E.B. da Silva and M.E. Romero, “Field Programmable Gate Array (FPGA) Based Neural Network Implementation of Stator Flux Oriented Vector Control of Induction Motor Drive”, IEEE International Conference on Industrial Technology, 2006.

[6] Xi Chen1, Gaofeng Wang, Wei Zhou, Sheng Chang, and Shilei Sun, “Efficient Sigmoid Function for Neural Networks Based FPGA Design”, ICIC 2006, LNCS 4113, pp. 672 – 677, Springer-Verlag Berlin Heidelberg 2006.

[7] M. Ghariani, M.W. Kharrat, N. Masmoudin, L. Kamoun, “Electronic implementation of a Neural Observer in FPGA technology: application to the control of electric vehicle”, 16th International Conference on Microelectronics, 2004.

[8] M. Qian, “Application of CORDIC Algorithm to Neural Networks VLSI Design”, IMACS Multiconference on Computational Engineering in Systems Applications, 2006.

[9] E. Soria-Olivas, J. D. Martín-Guerrero, G. Camps-Valls, A. J. Serrano-López, J. Calpe-Maravilla, and L. Gómez-Chova, “A Low-Complexity Fuzzy Activation Function for Artificial Neural Networks”, IEEE Transactions on Neural Networks, Vol. 14, No. 6, 2003.

[10] A. Rosado-Muñoz, E. Soria-Olivas, L. Gomez-Chova and J. Vila Francés, “An IP Core and GUI for Implementing Multilayer Perceptron with a Fuzzy Activation Function on Configurable Logic Devices”, Journal of Universal Computer Science, vol. 14, no. 10, 1678-1694, 2008.

[11] H. K. Kwan, “Simple Sigmoid-like Activation Function Suitable for Digital Hardware Implementation”, Electronics Letters, 1379-1380, 1992.

[12] R. W. Duren, R. J. Marks II, P. D. Reynolds and M. L. Trumbo, “Real-Time Neural Network Inversion on the SRC-6e Reconfigurable Computer”, IEEE Transactions on Neural Networks, Vol. 18, No. 3, May 2007.

[13] R. Andraka, A Survey of CORDIC Algorithms for FPGA Based Computers, Proc. of the ACM/SIGDA 6th Int. Symp. on FPGA, Monterey, CA, USA, pp. 191–200, 1998.

[14] T. Stieglitz and J. Meyer , Biomedical Microdevices for Neural Implants,Vol. 16 (BIOMEMS Microsystems, 2006), pp.71-137.

[15] J. Frounchi, G. Karimian, A.Keshtkar, An Artificial Neural Network Hardware for Bladder Cancer, European Journal of Scientific Research, Vol.27 No.1 (2009), pp.46-55.

[16] C. Lindsey, T. Lindblad, Survey of neural network hardware, In: Rogers, S.K., Ruck, D.W. (Eds.), Applications and Science of Artificial Neural Networks, SPIE 2492, 1194–1205, 1995.

[17] Y. Liao, Neural networks in hardware: a survey, available in the internet at http://wwwcsif.cs.ucdavis.edu/~liaoy/research/NNhardware.pdf (accessed February 2012).

[18] K. Hassibi, Detecting Payment Card Fraud with Neural Networks, Business Applications of Neural Networks, Progress in Neural Processing 13, Ed. P. Lisboa, B. Edisbury, A. Vellido, World Scientific, 2000.

[19] F.M.Dias, A.Antunes,A.M.Mota, Artificial neural networks: a review of commercial hardware, Engineering Applications of Artificial Intelligence, 17 (2004)945–952.

[20] J. Misra, I. Saha, Artificial neural networks in hardware: A survey of two decades of progress, Neurocomputing 74(2010)239–255.

Manuscript received November 24, 2011 Revised version received ….

Ivo M. Nascimento received his degree in Engineering Electronics and Telecommunications in 2009 at the University of Madeira and is currently finishing a Master's in Engineering Telecommunications and Networks at the University of Madeira in collaboration with INESC Porto.

Ricardo Jardim is currently pursuing a Master's in Engineering Telecommunications and Networks at the University of Madeira. His current research interests include Artificial Neural Networks, Artificial Intelligence, Biosensors for Monitoring Athletes Vital Signals and Heart Condition Patients and High Performance Computing.

Fernando Morgado-Dias received his *Diplôme D'Études Approfondies* in Microelectronics from the University Joseph Fourier in Grenoble, France in 1995 and his PhD from the University of Aveiro, Portugal, in 2005 and is currently an Assistant professor at the University of Madeira and Pro-Rector.
His research interests include Artificial Neural Networks and their applications, especially regarding their hardware implementations.