A Software Tool for Automatic Generation of Neural Hardware

Leonardo Reis, Luis Aguiar, Darío Baptista, and Fernando Morgado-Dias
Madeira Interactive Technologies Institute and Competence Centre for Exact Sciences and Engineering, University of Madeira, Portugal

Abstract: *Natural neural networks greatly benefit from their parallel structure that makes them fault tolerant and fast in processing the inputs. Their artificial counterpart, artificial neural networks, proved difficult to implement in hardware where they could have a similar structure. Although, many circuits have been developed, they usually present problems regarding accuracy, are application specific, difficult to produce and difficult to adapt to new applications. It is expected that developing a software tool that allows automatic generation of neural hardware while using high accuracy solves this problem and make artificial neural networks a step closer to the natural version. This paper presents a tool to respond to this need: A software tool for automatic generation of neural hardware. The software gives the user freedom to specify the number of bits used in each part of the neural network and programs the selected FPGA with the network. The paper also presents tests to evaluate the accuracy of the implementation of an automatically built neural network against Matlab.*

Keywords: *Artificial neural networks, feedforward neural networks, system generator, matlab, xilinx, simulink, integrated software environment.*

Received January 4, 2012; accepted August 26, 2012; published online April 4, 2013

1. Introduction

Natural Neural Networks are highly connected and largely parallel structures capable of performing several tasks that are very difficult to implement in a computer. The artificial imitation, Artificial Neural Networks (ANN), are structures composed of simplified neurons, connected in networks with a certain degree of parallelism that cannot be achieved in the sequential operation of a computer. As pointed out by [5]. The greatest potential of neural networks remains in the high-speed processing that could be provided through massively parallel VLSI implementations.

To achieve this parallelism a hardware implementation is needed. Nevertheless the number of implementations present in the literature and their capacity for being generic is very low. This situation can be changed if a simple and fast alternative for implementing ANNs in hardware is supplied [1].

A few attempts have been made in building such solutions [6, 10] but the proposed implementations are still very simple. In [6], we can find very simple blocks with low resolution and oversimplified activation functions. In [13], though using only Heaviside functions, we find a generic Simulink block for a neuron that can be translated by system generator. The most promising proposal found in the literature is in [10]. In this paper an IP Core is proposed for building synthesizable VHDL code for ANN but it only uses a simplified fuzzy prepared activation function that reduces the precision.

More work has been done regarding the manual implementation of ANN. The difficulties for these implementations are well known: the non-linearity of the hyperbolic tangent; the number of bits necessary to obtain high precision; the difficulties of using floating or the limitations of fixed point notation and the resources needed to implement a true parallel solution.

The implementation of the hyperbolic tangent as activation function received a large share of the attention in this area. The best solutions can be found in [4], where the Mean Square Error (MSE) in the hyperbolic tangent is 2.18×10^{-5} and in a control loop is 5×10^{-8}; in [2], the solution is based in a Taylor series which achieved an error of 0.51%; in [3], a piecewise linear implementation is proposed which obtained 0.0254 of "standard deviation with respect to the analytic form," in [11], a set of five polynomial 5th order approximations is proposed for a maximum error of 8×10^{-5}, using the sigmoid function. It should be noted that these results are hardly comparable but they are presented here as they were published.

In this paper we present a software tool for automatic generation of neural hardware: Automatic Neural Generator (ANGE). This software prepares and downloads the ANN to an FPGA with the characteristics defined by the user. ANGE tool uses Matlab and Simulink (from Mathworks) and Integrated Software Environment (ISE). Matlab and Simulink tools are widely used in the ANN field and ISE and System Generator are tools from the Xilinx Company to work with programmable hardware which is the preferred solution for a large part of the ANN implementations due to the acceptable price for a reduced number of copies [7]. The initial part of this work was presented in [1, 9].

The rest of the paper is organized as follows: section 2 describes ANN structure; section 3 describes shortly the neuron implementation developed; section 4 introduces the main tool; section 5 shows some of the results obtained; section 6 draws the conclusions and section 6 points the directions for future work.

2. ANN Structure

In this work we focus only in ANN of the feed forward type as shown in Figure 1, where there are no lateral nor feedback connections.

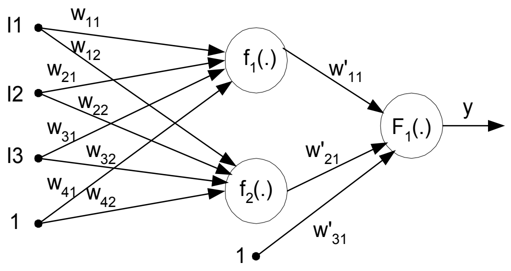


Figure 1. Example of a feedforward ANN.

A Feedforward Neural Network (FNN) is a layered structure, which can include non-linearity [12]. The basic element of a FNN is the neuron that is shown schematically in Figure 2.

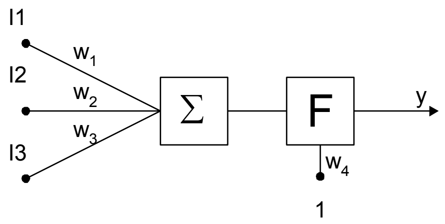


Figure 2. Neuron structure.

The neuron implements the general equation:

$$y = F(\sum_{i=1}^{n} Ii.wi) \tag{1}$$

Where usual functions for F are sigmoidal, linear and hard limit.

A FNN is composed of an input layer, one or more hidden layers with one or more neurons and an output layer where frequently the neurons are linear.

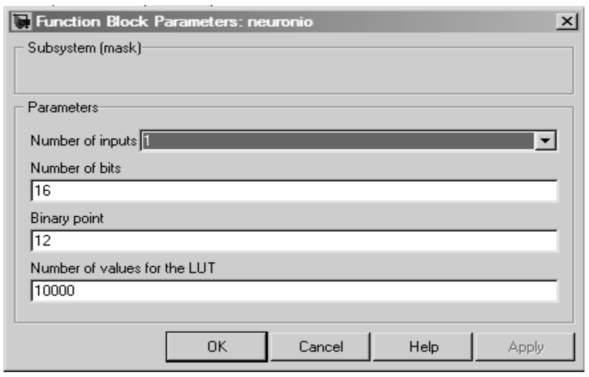
The multi input single output FNN in Figure 1 implements the following general equation:

$$y = F_1(\sum_{j=1}^{nh} w'_{j1} f_j(\sum_{l=1}^{nI} w_{lj} I_l)) \tag{2}$$

3. Neuron Implementation

The objective of this part of the work is to provide a way to implement a neuron in an automated way. We have chosen to do this using simulink and system generator. This allows designing blocks and test them with a friendly interface and, in the last stage, download and test the developed solutions within a FPGA. As a consequence of using system generator the circuit will be implemented in fixed point notation.

The neuron is created as a subsystem that can be configured using a mask. The mask used for our neuron block can be seen in Figure 3.


Function Block Parameters: neuronio

Subsystem (mask)

Parameters

Number of inputs: 1

Number of bits: 16

Binary point: 12

Number of values for the LUT: 10000

Figure 3. Mask for selecting the neuron parameters.

The parameters used in this mask are: the number of inputs, number of bits for the inputs, the position for the fixed point and the number of values for the Look Up Table (LUT) that implements the hyperbolic tangent activation function.

This mask hides the Matlab code that is responsible for placing the components in a simulink model and connect them in order to complete the desired system. An example of a simulink model for a neuron with four inputs can be seen in Figure 4.

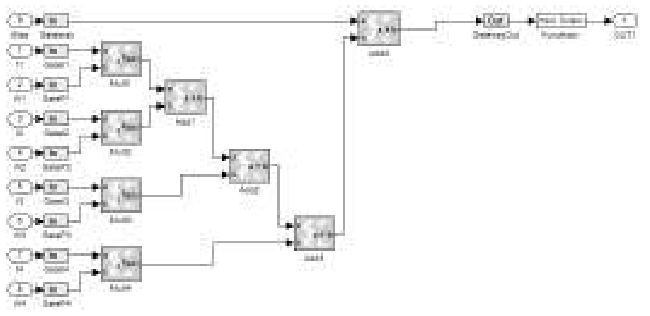
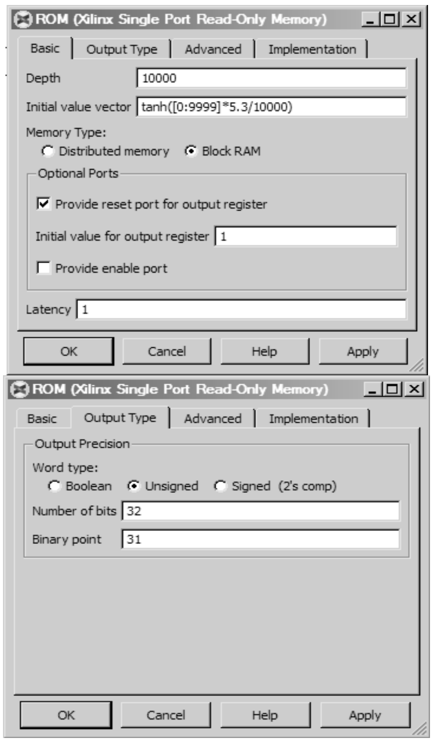


Figure 4. Simulink model for a neuron with four inputs.

Most of the blocks in this figure are common blocks. The FuncAtiv block holds the LUT that represents part of the hyperbolic tangent. Since it is an odd function, this block transforms the partial implementation in order to supply all the values necessary.

The configuration windows of the ROM that implements the LUT can be seen in Figure 5. The upper part shows a ROM with 10000 values and the choice of the values that fill the memory. The lower part shows the use of 32 bits, with only one for the integer part.


ROM (Xilinx Single Port Read-Only Memory)

Basic | Output Type | Advanced | Implementation

Depth: 10000

Initial value vector: tanh([0:9999]*5.3/10000)

Memory Type: ( ) Distributed memory (•) Block RAM

Optional Ports

[✓] Provide reset port for output register

Initial value for output register: 1

[ ] Provide enable port

Latency: 1

OK | Cancel | Help | Apply

ROM (Xilinx Single Port Read-Only Memory)

Basic | Output Type | Advanced | Implementation

Output Precision

Word type: ( ) Boolean (•) Unsigned ( ) Signed (2's comp)

Number of bits: 32

Binary point: 31

OK | Cancel | Help | Apply

Figure 5. Configuration of the ROM that implements the LUT.

- *Neuron Tests*

To analyse the precision of the solution developed a neuron with a single input, represented in Figure 6, was tested against the Matlab hyperbolic tangent function.

The subsystem block was connected as shown in Figure 7 so that an input of 100000 points was supplied to compare with the Matlab implementation of the hyperbolic tangent.

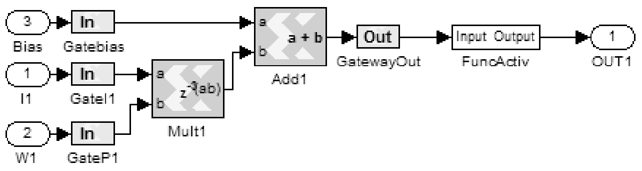


Figure 6. Single input neuron.

Using this topology several tests were performed such as the one shown in Figure 8. These tests include different number of bits for the input, different number of points in the ROM and different number of bits for each value stored in the ROM.

The different format for inputs and outputs should be understood by the fact that the range of values for each situation is different. The choice made here was to try to use only the number of bits that are really necessary for the integer part to implement the maximum value.

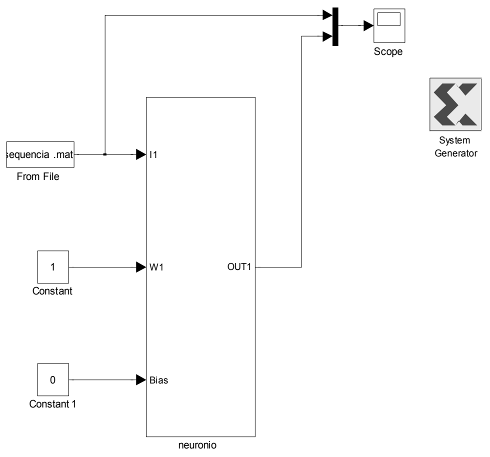


Figure 7. Single input neuron subsystem connected.

The example of Figure 8 is the one with fewer values in the LUT and with less number of bits used as input and output. It was chosen because it is the only one where a small difference between Matlab and LUT can be seen.

Table 1 shows MSE for these tests, where the format is marked as (total number of bits, number of bits of the decimal part).

Also, interesting is the distribution of the error against the input values. This is shown for one of the examples in Figure 9.

As can be seen from Figure 9, the error is larger in the regions where the slope of the hyperbolic tangent is steeper. This analysis makes us propose a test with the hyperbolic tangent proposed in [6], which we will do as further work.

Considering Table 1, the values of the error introduced seem to be low enough not to introduce drastic changes in the ANN behaviour if 16 or more bits are used with a minimum of 1000 points in the LUT though 5000 would be preferable. We must not forget that the error introduced in each neuron is propagated to other neurons in the network and can cause a larger change than expected.

Table 1. Summary of the tests performed to test the LUT's input, output, and dimension.

LUT Size	**MSE Input Format (8,4)**		**MSE Input Format (16,12)**		**MSE Input Format (32,28) Output Format (32,30)**
	Output Format (8,6)	**Output Format (32,30)**	**Output Format (16,14)**	**Output Format (32,30)**	
100	$4{,}7763\times10^{-5}$	$3{,}9175\times10^{-5}$	$1{,}6488\times10^{-5}$	$1{,}6488\times10^{-5}$	$1{,}6483\times10^{-5}$
500	$2{,}8570\times10^{-5}$	$2{,}2453\times10^{-5}$	$9{,}0789\times10^{-7}$	$9{,}0715\times10^{-7}$	$9{,}0794\times10^{-7}$
1000	$2{,}8362\times10^{-5}$	$2{,}1960\times10^{-5}$	$2{,}1837\times10^{-7}$	$2{,}1811\times10^{-7}$	$2{,}1737\times10^{-7}$
5000	$2{,}8322\times10^{-5}$	$2{,}1706\times10^{-5}$	$1{,}0574\times10^{-8}$	$1{,}0434\times10^{-8}$	$1{,}0103\times10^{-8}$
10000	$2{,}8322\times10^{-5}$	$2{,}1704\times10^{-5}$	$2{,}3817\times10^{-9}$	$2{,}2174\times10^{-9}$	$1{,}8837\times10^{-9}$

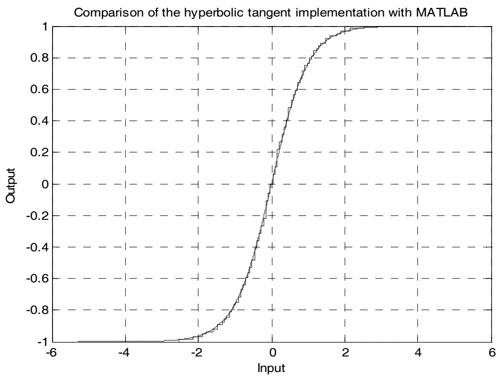


Figure 8. Hyperbolic tangent implementation using 100 points with 4 bits for the integer part and 4 bits for the decimal part.

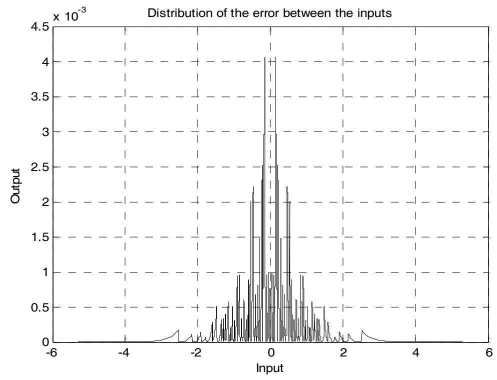


Figure 9. Distribution of the error against the input values.

4. The Automatic Generation Tool-ANGE

ANGE, in version 1.0, is prepared to work with multi-layer perceptron or feedforward neural networks with linear activation functions or hyperbolic tangents.

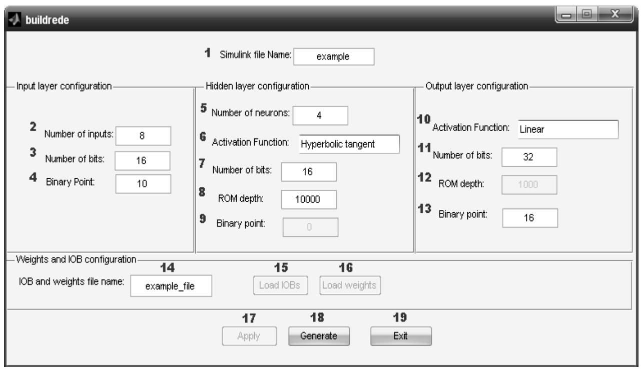
The hyperbolic tangent is implemented in its simplest way, although trying to maximize its performance and minimize the error obtained: using a LUT and reduced to the smallest part that can be used to represent the whole function.

ANGE runs over Matlab R2007b, with system generator 10.1 and ISE 10.1 and is capable of configuring hardware in a Field Programmable Gate Array (FPGA) for an ANN as large as the FPGA available allows. ANGE main window is in Figure 10.

As can be seen from Figure 10, the number of bits for the inputs, outputs and activation function can be selected by the user to accommodate the needs and capacity of the hardware available, using fields 3, 7 and 11. The position of the binary point (system generator uses fixed point) can also be selected, using 4, 9, and 13, in order to maximize the number of bits available after the point to increase the resolution.

The weights can be uploaded to configure all the network at once and it is also possible to upload information about which of the Input/Output Blocks (IOB) to use and what to connect to each of them, providing the file name in 14 and pushing 15, 16 or both.

After selecting the configuration and characteristics of the network, ANGE will automatically generate a simulink model file, similar to the one represented in Figure 11. The large blocks represented in this figure are the neurons, the inputs are identified by the word "In" and the weights are introduced using the constant blocks. As can be seen the ANN is implemented using full neurons and has a full parallel structure.



buildrede

1 Simulink file Name: example

Input layer configuration
2 Number of inputs: 8
3 Number of bits: 16
4 Binary Point: 10

Hidden layer configuration
5 Number of neurons: 4
6 Activation Function: Hyperbolic tangent
7 Number of bits: 16
8 ROM depth: 10000
9 Binary point: 0

Output layer configuration
10 Activation Function: Linear
11 Number of bits: 32
12 ROM depth: 1000
13 Binary point: 16

Weights and IOB configuration
14 IOB and weights file name: example_file

Figure 10. ANGE main window.

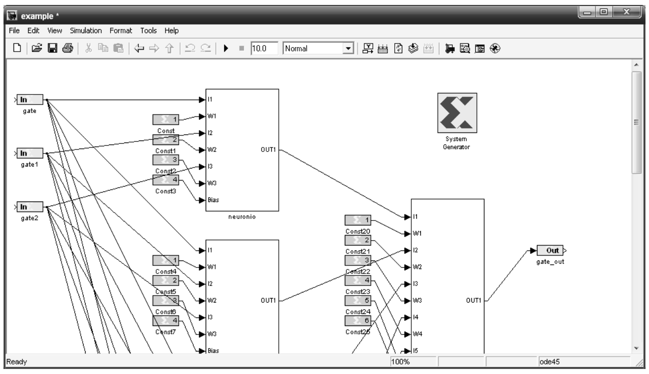


Figure 11. Example of an ANN generated by ANGE with the weights loaded.

ANGE can also be used to create co-simulation blocks. These blocks can be inserted in a Simulink library and used in Simulink models, as shown in Figure 12, inserting the FPGA in the loop and approaching the simulation to the real functioning of the system.

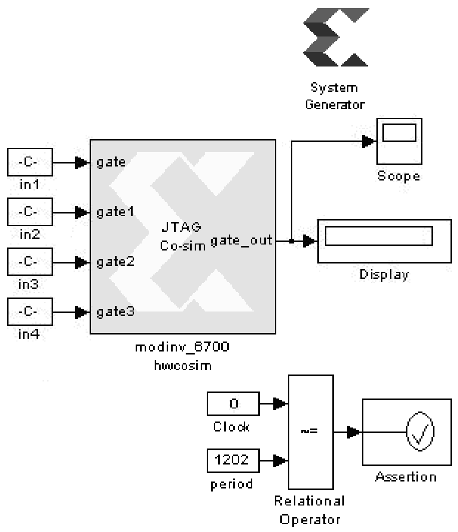


Figure 12. Example of an ANN generated by ANGE used as a co-simulation block.

5. Results Obtained

The ANN models must be inserted in a system to be tested. To test ANGE's implementation and evaluate the error introduced by its fixed point notation, models from a reduced scale kiln were used. This kiln was prepared for the ceramic industry and further details can be seen in [11].

For these tests two sets of models that represent a direct and an inverse model of the system were used in order to construct a Direct Inverse Control (DIC) loop, as represented in Figure 13.

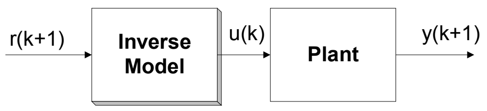


Figure 13. Block diagram of the direct inverse control.

This kind of loop, though very simple and easy to understand, requires the models to have a very good match and therefore is the best loop to be used when testing hardware implemented models because if the implementation reduces the quality of the models it will be seen directly on the control results.

The two sets of models, though representing the same system, were trained under different conditions and are different in the number of neurons in the hidden layer.

The ANNs were implemented using 16 bits for the inputs and representation of the hyperbolic tangent and 32 bits for the output. The LUT that holds the hyperbolic tangent values contains 10000 values.

The models were tested using two reference signals and they compare a result obtained implementing both models in Matlab with another one where the inverse model is implemented in an FPGA, using co-simulation. Comparing the two DIC versions allows for an indirect measurement of the error introduced by the system generator implementation.

Some of the results can be seen in Figures 14 and 15 and they were measured in terms of Mean Square Error (MSE) and are summarized in Table 2.

As can be seen, the maximum error introduced in the hardware implementation of the models was of 15, 31%. This value is not very low but there are important aspects that should be mentioned: the control loop maintained stability and the error introduced seems to be almost constant and therefore represents a larger percentage when the error in Matlab is smaller. The error introduced results from using less bits, fixed point notation and a LUT to represent the hyperbolic tangent. Its maximum value can be derived by the number of bits truncated and the maximum step between consecutive values in the LUT. As a result the error introduced should be bounded and small and be reduced with the increase of the number of bits used in the solution.

In hardware implementations besides precision, it is important that the final solution does not use too many resources. To evaluate this, Table 3 shows a resume of the resources used in proportion to the capacity of the FPGA, a Virtex 5 5VFX30TFF665-2S. The ANNs used are not very big (25fmd has 4 inputs and 8 neurons, while fmdb1 has 4 inputs and 6 neurons) and the FPGA is a small Virtex 5, which means that with a more recent FPGA it is possible to implement an ANN more than 20 times larger than the ones used as example here.

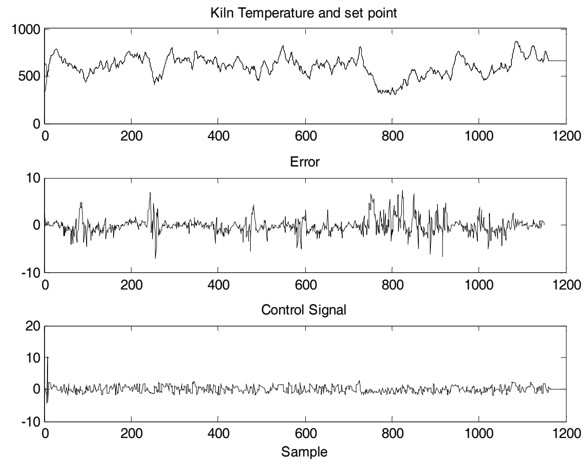


Figure 14. Results of model fmd1b with reference 1.

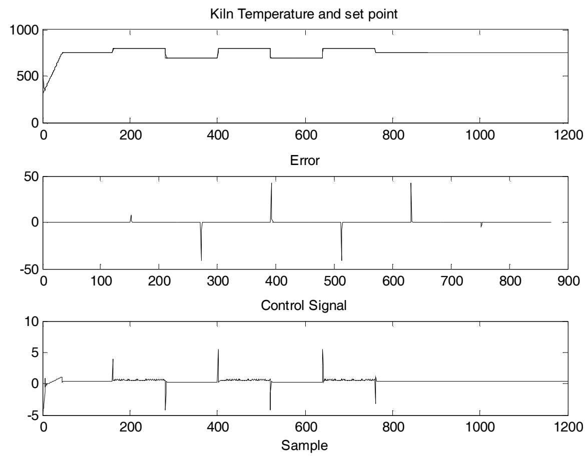


Figure 15. Results of model 25fmd with reference 2.

Table 2. Mean square error in co-simulation and Matlab.

Model and Reference	Co-simulation	Matlab	Error Change
Fmdb1 – Ref1	2,5155	2,5173	0,07%
Fmdb1 – Ref2	72,4492	72,3548	0,13%
25fmd – Ref1	0,01507	0,01307	15,31%
25fmd – Ref2	8,1688	8,1542	0,18%

Table 3. Resume of the resources used in the implementation of both neural models for LUT with 10000 values.

Resources	RAM16 % of 68	Registers % of 20480	LUTS % of 20480	DSP48ES % of 64
25fmd	58	22	31	62
fmdb1	44	17	22	46

6. Conclusions

This paper presents ANGE, a tool for automatic generation of neural hardware and shows some of the results obtained with it.

ANGE makes use of system generator, which works only with fixed point notation. The use of fixed point notation and a simplified LUT representation of the hyperbolic tangent introduce an error that must be analyzed in order to see if it is acceptable.

Results of a test are presented with a small network with a medium resolution fitted in a small Virtex 5 FPGA. The control loop shown presents an acceptable error with medium resolution and keeps the loop stable. With larger number of bits (possible with ANGE's present version) and more accurate implementation of the hyperbolic tangent these results will be further improved.

ANGE will allow fast prototyping using the preferred hardware target for the neural community: FPGA. It is expected to contribute to a new growth of hardware implementations of ANN.

7. Further Work

The work presented, although represents an important step, can be improved specially regarding the activation function. The next version of ANGE will have more options for the implementation of the hyperbolic tangent.

Acknowledgments

The authors would like to acknowledge the Portuguese Foundation for Science and Technology for their support for this work through project PEst-OE/EEI/LA0009/2011.

References

[1] Aguiar L., Reis L., and Morgado-Dias F., "Neuron Implementation Using System Generator," *in Proceedings of the 9th Portuguese Conference on Automatic Control*, Portugal, 2010.

[2] Arroyo M., Ruiz A., and Leal R., "An Artificial Neural Network on a Field Programmable Gate Array as a Virtual Sensor," *in Proceedings of the Third International Workshop on Design of Mixed-Mode Integrated Circuits and Applications*, Mexico, pp. 114-117, 1999.

[3] Ayala J., Lomeña A., López-Vallejo M., and Fernández A., "Design of a Pipelined Hardware Architecture for Real-Time Neural Network Computations," *in Proceedings of IEEE Midwest Symposium on Circuits and Systems*, USA, pp. 419-422, 2002.

[4] Ferreira P., Ribeiro P., Antunes A., and Morgado-Dias F., "A High Bit Resolution FPGA Implementation of a FNN with a New Algorithm for the Activation Function," *Neurocomputing*, vol. 71, no. 1-3, pp. 71-77, 2007.

[5] Lippmann R., "An Introduction to Computing with Neural Nets," *IEEE ASSP Mag*, vol. 4, no. 22, pp. 4-22, 1987.

[6] Moctezuma-Eugenio J. and Huitzil C., "Estudio Sobre la Implementación de Redes Neuronales Artificiales Usando XILINX System Generador," *in Proceedings of XII Workshop Iberchip*, Costa Rica, pp. 1-7, 2006.

[7] Morgado-Dias F., Antunes A., and Mota A., "Artificial Neural Networks: a Review of Commercial Hardware," *Engineering Applications of Artificial Intelligence-IFAC*, vol. 17, no. 8, pp. 945-952, 2004.

[8] Morgado-Dias F. and Mota A., "Direct Inverse Control of a Kiln," *in Proceedings of the 4th Portuguese Conference on Automatic Control*, pp. 336-341, 2000.

[9] Reis L., Aguiar L., Baptista D., and Morgado-Dias F., "ANGE-Automatic Neural Generator," *in Proceedings of the 21st International Conference on Artificial Neural Networks*, Berlin, Germany, pp. 446-453, 2011.

[10] Rosado-Muñoz A., Soria-Olivas E., Gomez-Chova L., Vila J., "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, pp. 1678-1694, 2008.

[11] Soares A., Pinto J., Bose B., Leite L., Da-Silva L., Romero M., "Field Programmable Gate Array (FPGA) Based Neural Network Implementation of Stator Flux Oriented Vector Control of Induction Motor Drive," *in Proceedings of IEEE International Conference on Industrial Technology*, Mumbai, pp. 31-34, 2006.

[12] Taspinar N. and Isik Y., "Multiuser Detection with Neural Network MAI Detector in CDMA Systems for AWGN and Rayleigh Fading Asynchronous Channels," *the International Arab Journal of Information Technology*, vol. 10, no. 4, pp. 413-419, 2012.

[13] Tisan A., Buchman A., and Oniga S., "A Generic Control Block for Feedforward Neural Network with On-Chip Delta Rule Learning Algorithm," *in Proceedings of the 30th International Spring Seminar on Electronics Technology*, Cluj-Napoca, pp. 567-570, 2007.

Leonardo Reis received his bachelor's degree in electronics and telecommunications engineering from the University of Madeira, Portugal in 2011, and is currently finishing his Telecommunications and Energy Networks Master's thesis at the same university.

Luís Aguiar finished his bachelor's degree in electronics and telecommunications engineering in 2009 and Master's degree in Telecommunications and Networks Engineering in 2011, both at the University of Madeira. He is currently researcher at the University of Madeira, Portugal.

Darío Baptista received his bachelor's degree in electronics and telecommunications in 2007 and his Master's degree in telecommunications and network engineering in 2009 both from the University of Madeira, Portugal. His main research field is artificial neural networks and he is currently researcher at the University of Madeira.

Fernando Morgado-Dias received his Master's degree 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 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.