ANGE – Automatic Neural Generator

Leonardo Reis[1], Luis Aguiar[1], Darío Baptista[1], Fernando Morgado Dias[1]

[1] Centro de Competências de Ciências Exactas e da Engenharia, Universidade da Madeira – Campus penteada, 9000-039 Funchal, Madeira, Portugal

Abstract. Artificial Neural Networks have a wide application in terms of research areas but they have never really lived up to the promise they seemed to be in the beginning of the 80s. One of the reasons for this is the lack of hardware for their implementation in a straightforward and simple way. This paper presents a tool to respond to this need: An Automatic Neural Generator. The generator allows a user to specify the number of bits used in each part of the neural network and programs the selected FPGA with the network. To measure the accuracy of the implementation an automatically built neural network was inserted in a control loop and compared with Matlab.

Keywords: Artificial Neural Networks, Feedforward Neural Networks, System Generator, Matlab, Xilinx, Simulink, Integrated Software Environment.

1 Introduction

Artificial Neural Networks (ANN) are structures composed of neurons, connected in networks with massive parallelism that can greatly benefit from hardware implementation. To benefit from this parallelism a hardware implementation is needed but 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 [2-4] but the proposed implementations are still very simple. In [2] we can find very simple blocks with low resolution and oversimplified activation functions. In [3], 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 [4]. 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 choice of using floating or fixed 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 [5], where

the maximum error is $5x10^{-8}$ in a control loop; in [6] the solution is based in a Taylor series which achieved an error of 0,51%; in [7] a piecewise linear implementation is proposed which obtained 0.0254 of "standard deviation with respect to the analytic form", in [8] a set of five polynomial 5^{th} order approximations is proposed for a maximum error of $8x10^{-5}$, using the sigmoid function.

In this paper we present an Automatic Neural GEnerator (ANGE) that programs ANNs in an FPGA with the characteristics defined by the user. The ANGE tool uses Matlab and Simulink (from Mathworks) and Integrated Software Environment (ISE), also from Xilinx. Matlab and Simulink tools are widely used in the ANN field and the 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 [9]. The initial part of this work was presented in [1].

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

2 Neuron implementation

The first step towards the automatic implementation of ANN was obtained implementing the neuron. Using the tools mentioned in the previous section, the neuron is configured with a mask that is represented in figure 1.

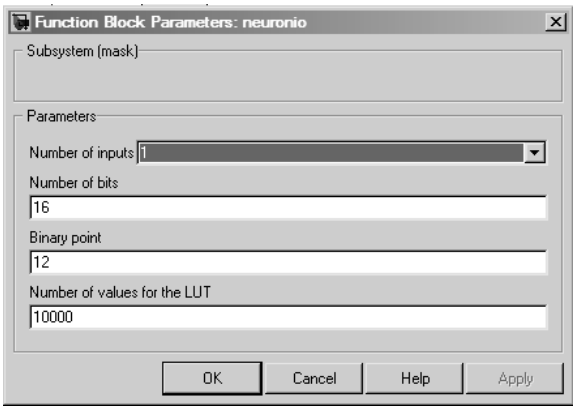

Fig. 1. Mask for selecting the neuron parameters.

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 2.

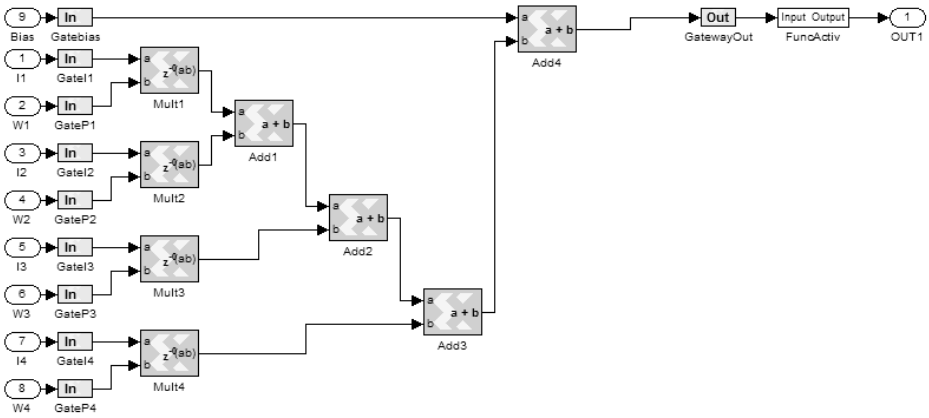

Fig. 2. Simulink model for a neuron with four inputs.

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

The configuration windows of the ROM that implements the LUT can be seen in figure 3. The left side shows a ROM with 10000 values and the choice of the values that fill the memory. The right side shows the use of 32 bits, with only one for the integer part.

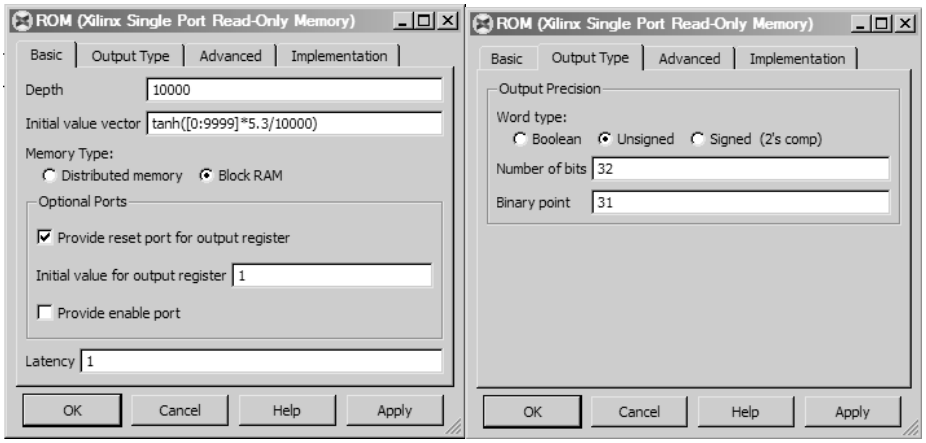

Fig. 3. Configuration of the ROM that implements the LUT.

3 ANGE details

ANGE, for its first version, 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 though 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 4.

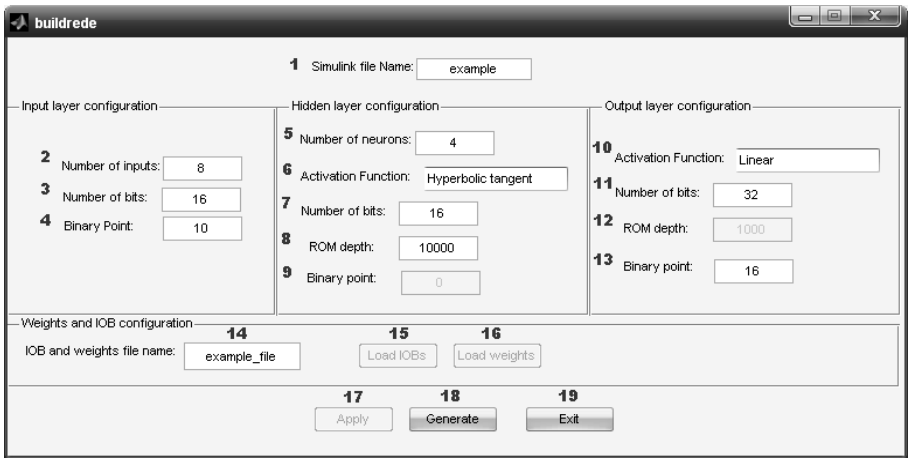

Fig. 4. ANGE main window.

As can be seen, 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 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 5. 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.

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 6, inserting the FPGA in the loop and approaching the simulation to the real functioning of the system.

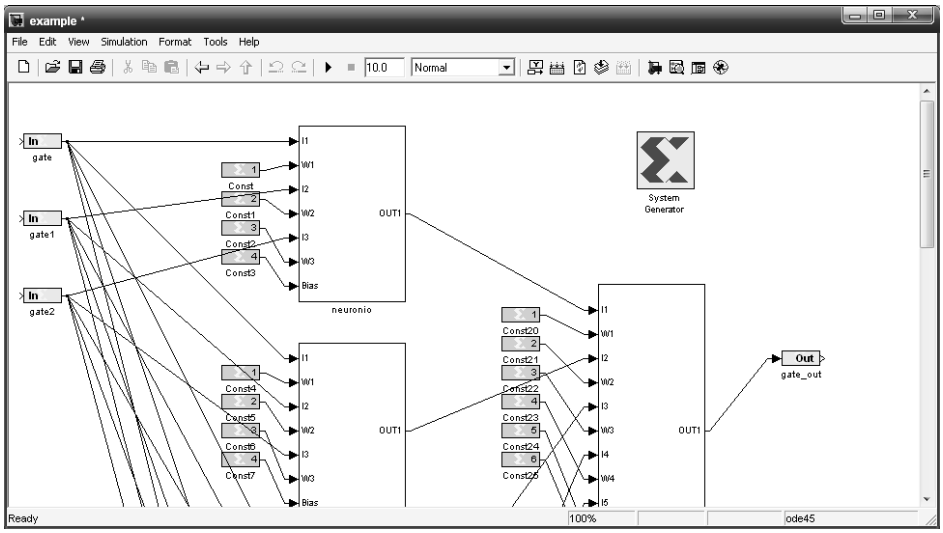

Fig. 5. : Example of an ANN generated by ANGE with the weights loaded.

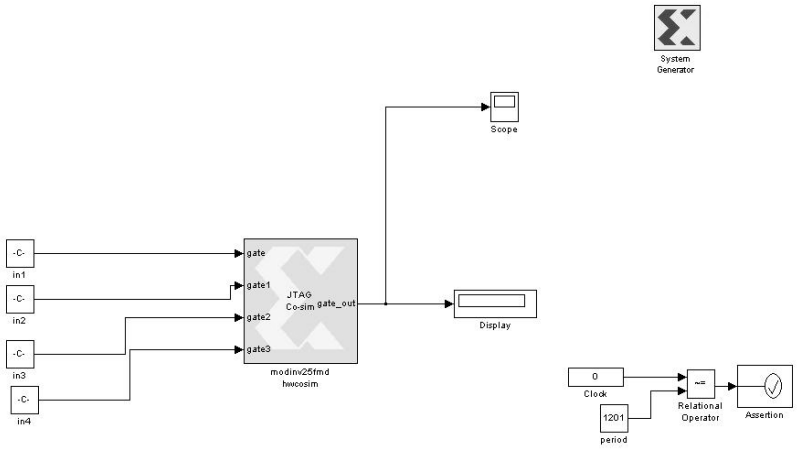

Fig. 6. : Example of an ANN generated by ANGE used as a co-simulation block.

4 Test results

ANN models serve a purpose only when they represent some system or behaviour. 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 [10].

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 loop, as represented in figure 7.

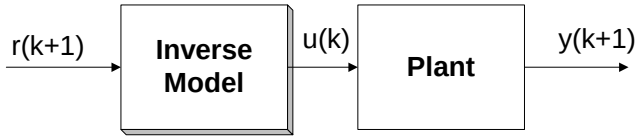

Fig. 7. : Block diagram of the Direct Inverse Control.

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. Some of the results can be seen in figure 8 and they were measured in terms of Mean Square Error (MSE) and are summarized in table 1.

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 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.

Table 1. 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%

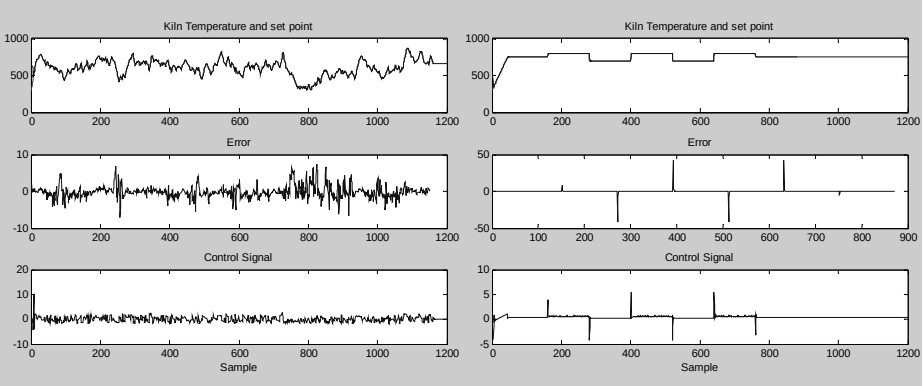

Fig. 8. : Results of model fmd1b with reference 1 and model 25fmd with reference 2.

In hardware implementations besides precision it is important that the final solution does not use too many resources. To evaluate this, table 2 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.

Table 2. 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

5 Conclusion

This paper presents ANGE, an Automatic Neural Hardware Generator and shows some of the results obtained with it.

A test is presented with a small network with a medium resolution fitted in a small Virtex 5 FPGA. The control loop shown presents 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 and is expected to contribute to a new growth of hardware implementations of ANN.

6 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.

References

1. Aguiar, L., Reis, L., Morgado Dias, F.: Neuron Implementation Using System Generator, 9th Portuguese Conference on Automatic Control, 2010.
2. Moctezuma Eugenio, J. C., Huitzil, C. T.: Estudio Sobre la Implementación de Redes Neuronales Artificiales Usando XILINX System Generador, XII Workshop Iberchip, Costa Rica, 2006.
3. Tisan, A., Buchman, A. Oniga, S. Gavrincea: A Generic Control Block for Feedforward Neural Network with On-Chip Delta Rule Learning Algorithm, C., North Univ. of Baia Mare, Baia Mare, 30th International Spring Seminar on Electronics Technology, 2007.
4. Rosado-Muñoz, A., Soria-Olivas, E., Gomez-Chova, L., Vila Francés, 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, 1678-1694, 2008.
5. Ferreira, P., Ribeiro, P., Antunes, A., Morgado Dias,F.: 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, 2007.
6. Arroyo Leon, M. A., Ruiz Castro, A., Leal Ascenccio, R.R.: 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.
7. Ayala, J. L., Lomeña, A. G., López-Vallejo, M., Fernández, A.: Design of a Pipelined Hardware Architecture for Real-Time Neural Network Computations, IEEE Midwest Symposium on Circuits and Systems, USA, 2002.
8. Soares, A.M., Pinto, J.O.P., Bose, B.K., Leite, L.C., da Silva, L.E.B., Romero, M.E.: 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.
9. Morgado Dias, F., Antunes, A., Mota, A.: Artificial Neural Networks: a Review of Commercial Hardware, Engineering Applications of Artificial Intelligence, IFAC, Vol. 17/8, pp. 945-952, 2004.
10. Morgado Dias, F., Mota, A.: Direct Inverse Controlo f a Kiln, 4th Portuguese Conference on Automatic Control, 2010.