FPGA Implementation of a NARX Network for Modeling Nonlinear Systems

J.A. Rentería-Cedano[1], L.M. Aguilar-Lobo[1], S. Ortega-Cisneros[1], J.R. Loo-Yau[1], and Juan J. Raygoza-Panduro[2]

[1] Departamento de Ing. Eléctrica y Ciencias Computacionales
Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara, Av. Del Bosque 1145, Colonia El Bajío,
C.P. 45019, Zapopan, Jalisco, México
[2] Departamento de Electrónica
Centro Universitario de Ciencias Exactas e Ingenierías
Universidad de Guadalajara
{jrenteria,laguilar}@gdl.cinvestav.mx

Abstract. This paper presents the FPGA implementation of a NARX neural network for the modeling nonlinear systems. The complete neural architecture was implemented with Verilog language in Xilinx ISE Tool with the Virtex-6 FPGA ML605 Evaluation Kit. All operations, such as data processing, weight connections, multipliers, adders and activation function were performed using floating point format, because allows high precision in operations with high complexity. Some resources of Xilinx were used such as multipliers and CORE blocks, and the hyperbolic tangent of the activation is realized based on Taylor series. To validate the implementation results, the NARX network was used to model the inverse characteristics of a power amplifier. The results obtained in the simulation and the FPGA implementation shown a high correspondence.

Keywords: FPGA implementation, neural network, nonlinear behavior, Xilinx, floating point, Taylor series, CORDIC.

1 Introduction

Nowadays the Neural Networks (NN) are used in problems that involve nonlinear behavior. A high number of diverse NN topologies have been development for optimize the performance in nonlinear modeled applications. In this context, the NARX network is one of the NN more precise for model nonlinear behavior. The NARX network is a Recurrent Neural Network (RNN) with tapped delay lines in the input and output, which allows modeling the short and long-term dependencies [1]. We are using the NARX network to model a nonlinear system, such as, a power amplifier (PA). However, at this moment only has been development the neural architecture in simulation environment [2]. This work comprises the hardware implementation of the NARX network architecture as a modeled system to obtain the inverse characteristics of a PA.

Actually, the Field Programmable Gate Array (FPGA) has been used for several implementations of NN with different mathematically models, applications and hardware characteristics [3-5]. FPGA provides some advantages, such as rapid prototyping, adaptation, reduced costs and simplicity in the design. The hardware implementation of a NN in FPGA involving several aspects, such as, the data representation (fixed or floating point), operands word length, arithmetic operators characteristics (adder, multiplier, etc.), activation function, number of inputs and outputs of the network, and network structure (number of neurons, number of delay lines) [6].

This paper describes the FPGA implementation of a NARX network to model the inverse characteristics of a PA using Verilog language in Xilinx ISE Tool with the Virtex-6 FPGA ML605 Evaluation Kit [7]. To validate the implementation, the FPGA and simulation results are compared and show a high correlation.

2 NARX Neural Network

The NARX neural network is a RNN with tapped delay lines at the input and output of the system to be modeled and its response is given by the discrete-time Nonlinear Autoregressive with eXogenous inputs (NARX) system [1]. The neural architecture of the NARX network is shown in Fig. 1. This architecture is formed by three mains layers: input, hidden, and output layers. The input layer is a set of tapped delay lines in the input and output of the system, the hidden layer is a set of neurons with sigmoidal activation function, and the output layer is a set of neurons with linear activation function. Mathematically, the output in a NARX neural network is given by:

$$y(n)=\sum_{k=1}^{m} w_k^{LW} G_k(n)+b^2 \tag{1}$$

where

$$X_k(n)=\sum_{j=1}^{du} w_{(k,j)}^{IW} u(n-j)+\sum_{j=1}^{dy} w_{(k,j)}^{IW} y(n-j)+b_k^1 \tag{2}$$

and

$$G_k(n)=\tanh\left(X_k(n)\right). \tag{3}$$

The embedded memory characteristic of the NARX network represents a significant advantage with respect to other RNNs. These tapped delay lines at the network output helps to have convergence faster and generalize better that other RNNs. In addition, in problems that have long-term dependencies, the performances of the NARX network are much better than other conventional RNNs. The explanation for this behavior is that the output memories of the NARX network can be manifested as jump-ahead connections in the time unfolded network.

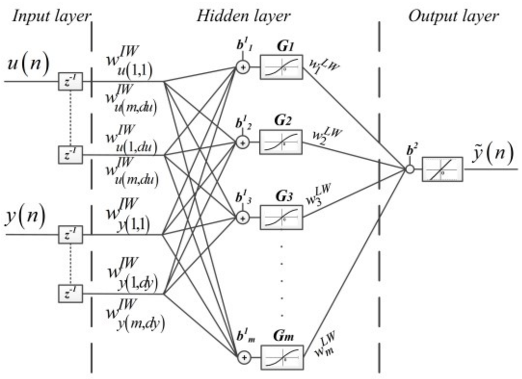

Fig. 1. Neural architecture of the NARX network

3 FPGA Implementation

The complete neural architecture of the NARX network was implemented in the Xilinx ISE Tool and is shown in Fig. 2. The implementation comprises all the operation of weights connections, multipliers, sums, and activations functions.

Some functions such as sums and divisions were implemented using Xilinx Floating-Point IP CORE. The control of the neural architecture was realized by a Finite State Machine (FSM) type Mealy machine [8]. The FSM is used instead of a processor because permit made a customer design without have an unnecessary recourses consume. The FSM has 20 states that perform 180 products, 192 sums, and 10 hyperbolic tangents, executed in 595 cycles.

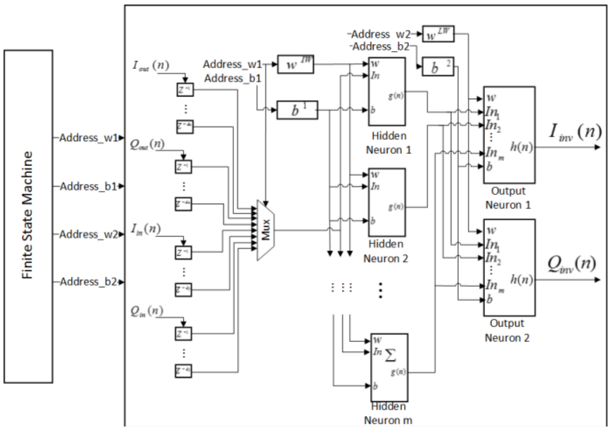

Fig. 2. Complete implemented architecture of the NARX Network

The neuron model is shown in a Fig. 3.a. and the components in each one are: inputs (I), weights (w), bias (b), activation function (f), and adder (+). The scheme used in the implementation was realized based in [9] and is shown in a Fig. 3.b. The basic operation realized in each neuron is initiated with the sample selection and it corresponding memory address in the weights matrix. Then, the multiplication of the connections between the inputs and weights are performed. The results are added and accumulated for all the connections. Next, the bias is added to each result and this value is passed through the activation function.

The multiplier block is implemented in floating point with simple precision and its output is given by [10]

$$Z = (-1)^{s1 \oplus s2}(1.mant1 * 1.mant2) * 2^{e1+e2} \tag{4}$$

where *s1* and *s2* are the signs, *mant1* and *mant2* are the representative values and finally *e1* and *e2* are the exponents containing the value of the power of the base.
Using hardware description language is possible to obtain a latency of 5 clock cycles to perform a multiplication, though the implementation of Xilinx Floating-Point IP CORE has a default latency of 8 clock cycles. By decreasing the latency resource usage increases.

The adder block is implemented with a Xilinx floating-point IP CORE and a register to store the results. The activation function used in the hidden layer is the hyperbolic tangent. There are several methods and algorithms to obtain the hyperbolic tangent, such as LUTs, CORDIC [11] and hybrids methods [12]. The Xilinx CORDIC IP block is not used because is required added a division to obtain the hyperbolic tangent. Therefore, we proposed a method to find the hyperbolic tangent based on Taylor series. The hyperbolic tangent is given by

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5}$$

The expansion of the exponential by the Taylor series can be written as:

$$e^{x} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \ldots + \frac{x^n}{n!} \tag{6}$$

Then, using (5) in (6) and performing algebraic operations was obtained the equivalent function to represent the hyperbolic tangent, such is given by

$$\tanh(x) = \frac{362880x + 60480x^3 + 3024x^5 + 72x^7 + x^9}{362880 + 181440x^2 + 15120x^4 + 504x^6 + 9x^8} \tag{7}$$

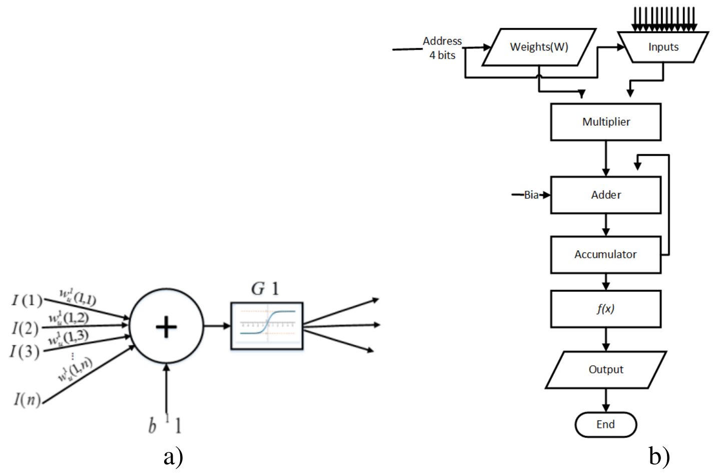

Fig. 3. a) Neuron model. b) Scheme of the single neuron model.

All operation using floating point format for the data representation used, because allows high precision in operations with high complexity. The floating point format is commonly represented in simple precision of 32 bits and double precision of 64 bits.

The training process of the NARX network was realized using the toolbox of Neural Networks (NN toolbox) of MATLAB [13], and the weights matrices are stored in a memory unit after the training process. The definition of the neural architecture parameters, such as, stop criterion [14], selected the memory order, number of neurons are realized before the training process to define the neural architecture. The NN toolbox include the pre-processing and post-processing in the input and output of the network respectively, that is showed in Fig. 4 and is given by:

$$y_{norm} = \frac{(y_{\max} - y_{\min}) * (x_{in} - x_{\min})}{(x_{\max} - x_{\min})} + y_{\min} \tag{8}$$

where y_{min} and $y_{\max}$ are the normalization values (+/-1), x_{min} and x_{max} are the maximum and minimum values of the vector and x_{in} is the actual data to process. Therefore this pre-processing and post-processing should be realized in the FPGA.

Finally, communication asynchrony (UART) [15] was realized to transfer the output data of the FPGA to PC to validate the results.

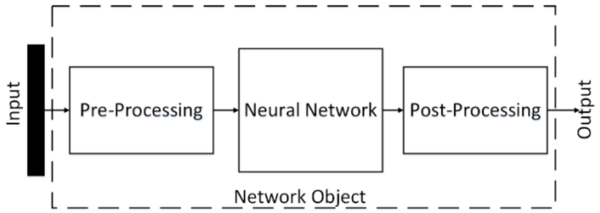

Fig. 4. Pre-processing and post-processing realized in the NN toolbox of Matlab

4 Validation and Results

An experimental setup to model the inverse characteristics of a PA is used to validate the FPGA implementation of the NARX network [2]. The validation is realized in three stages, in the first stage was realized the measurements of the input and output signals of the PA. The second stage includes the training of the NARX network and the final stage is the validation of the results.

First, the parameters of the NARX network are defined. The input vector is formed by four inputs that represent I (real) and Q (imaginary) components of the input and output signals of the PA. Four delay lines were selected for each one (input and output) and 10 neurons are selected in the hidden layer.

Table 1. Comparison between the FPGA implementation and MATLAB simulation results

Results	Real(FPGA)	Imag(FPGA)	Real(Matlab)	Imag(Matlab)
1	-0.34510845317838	0.07818963943093	-0.34510342853533	0.07818700472594
2	-0.35257565491600	-0.02967927526307	-0.35257891077726	-0.02967838693830
3	-0.24657515530991	-0.07807854950230	-0.24664802091774	-0.07809724509940
4	-0.06395308502237	-0.03039482902367	-0.06402700042977	-0.03042185011335
5	0.12606102178778	0.07410325482868	0.12605209565504	0.07410446812629
6	0.25096877400903	0.17675751911770	0.25096190357323	0.17675773608755
7	0.28196600764153	0.20763142923692	0.28196250978712	0.20762731818684
8	0.21321615559612	0.15994817431051	0.21321632703128	0.15994433789154
9	0.12060241480182	0.06695307107320	0.12060398021669	0.06695367249790
10	0.03242863820595	0.00092798982272	0.03242695709790	0.00092780150537

The neural architecture of the NARX network was training with the NN toolbox in MATLAB and the weights and biases matrices were stored in a RAM block. Then, the neural architecture obtained was implemented in a FPGA Virtex-6.

The results obtained after the communication between the FPGA and the PC are compared with the simulation results of MATLAB. Experimental results are shown in Table 1 and shown a high correlation between MATLAB and FPGA.

It can be seen that the differences between the implementation and simulation results only are present after the six decimal, and this is produced because the FPGA implementation is based on 32 bits and the MATLAB simulation employed 64 bits. The error calculated is less than 1%. Table 2 gives the resource requirements reported by Xilinx in details.

Table 2. Resource utilization of the Virtex-6

Resource utilization	Hidden Neuron	Output Neuron	Multiplier	Hyperbolic Function	Adder	Total Used	Available
Slice Registers	3699	672	97	3027	578	38572	301440
Slice LUTs	2697	573	96	2131	446	29057	150720
LUT FF	2116	461	79	1645	354	21491	89013
Bonded IOBs	137	135	100	68	68	3	600
Block RAM						225	416
DSP	6	2	2	4	2	64	768
Maximum Operation Frequency	95.511 MHz						

5 Conclusions

The FPGA implementation of the NARX neural network has been presented to model the inverse characteristics of a PA. The implementation results showed a high correlation with the MATLAB simulation. An error is presented in the less significant bits. Moreover a method to implement the hyperbolic tangent function by means of an expansion of the Taylor series is presented. This method can be more complex but has higher accuracy with respect to others methods. The NARX was implemented in a sequential form, a disadvantage with pipeline or segmented architectures. The validation results in nonlinear behavior modeled shown an efficient FPGA implementation.

References

1. Siefelmann, H.T., Horne, B.G., Giles, C.L.: Computational capabilities of recurrent NARX neural networks. IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics 27(2), 208–215 (1997)
2. Aguilar-Lobo, L.M., Garcia-Osoria, A., Loo-Yau, J.R., Ortega-Cisneros, S., Moreno, P., Rayas-Sanchez, J.E., Reynoso-Hernández, J.A.: A Digital Predistortion Technique Based on a NARX Network to Linearize GaN Class F Power Amplifiers. In: IEEE 57th International Midwest Symposium on Circuits And Systems (August 2014)
3. Bahoura, M., Park, C.-W.: FPGA-Implementation of an Adaptive Neural Network for RF Power Amplifier Modeling. In: 2011 IEEE 9th International New Circuits and Systems Conference (Newcas), pp. 29–32 (June 2011)
4. Atencia, M., Boumeridja, H., Joya, G., Garcia-Lagos, F., Sandoval, F.: FPGA Implementation of a Systems Identification Module Based Upon Hopfield Networks. Neurocomputing 70(2007), 2828–2835 (2007)
5. Bastos, J.L., Figueroa, H.P., Monti, A.: FPGA Implementation of Neural Networks-Based Controllers for Power Electronics Applications. In: Twenty-First Annual IEEE Applied Power Electronics Conference and Exposition, APEC 2006, pp. 1–6 (2006)
6. Braga, A.L.S., Llanos, C.H., Gohringer, D., Obie, J., Becker, J., Hubner, M.: Performance, Accuracy, Power Consumption and Resource Utilization Analysis for Hardware/Software realized Artificial Neural Networks. In: 2010 IEEE Fifth Internatiol Conference on Bio-Inspired Computing: Theories and Aplications (BIC-TA), pp. 1629–1636 (September 2010)
7. Virtex-6 FPGA ML605 Evaluation Kit, Xilinx Inc., `http://www.xilinx.com/products/boards-and-kits/EK-V6-ML-605-G.htm`
8. Salcic, Z., Smailagic, A.: Digital system design and prototyping using field programmable logic, pp. 134–141. Kluwer Academic Publishers, Boston (1997)
9. Mohamad, K., Mahmud, M.F.O., Adnan, F.H., Abdullah, W.F.H.: Design of single neuron on FPGA. In: 2012 IEEE Symposium on Humanities, Science and Engineering Research, SHUSER (2012)
10. IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2008), Revision of IEEE Std 754-4985 (August 29, 2008)
11. Qian, M.: Application of CORDIC Algorithm to Neural Networks VLSI Design. In: Multconf. Computational Engineering in Systems Applications IMACS, pp. 504–508 (October 2006)

12. Sartin, M.A., da Silva, A.C.R.: Approximation of Hyperbolic Tangent Activation Function Using Hybrid Methods. Department of Computing. UNEMAT- Universidade do Estado de Mato Grosso, Colider, MT, Brazil
13. Neural Network Toolbox, User's Guide R2014a, 2-9, 2-10, 2-11. The MathWorks.Inc., http://www.mathworks.com/help/pdf_doc/nnet/nnet_ug.pdf
14. Wang, W., Van Gelder, P.H.A.J.M., Vrijling, J.K.: Some issues about the generaliza-tion of neural networks for time series prediction
15. Wakhle, G.B., Aggarwal, I., Gaba, S.: Synthesis and Implementation of UART Using VHDL Codes. In: International Symposium on Computer, Consumer and Control, ICANN 2005. LNCS, vol. 3697, pp. 559–564 (2005)