2014 11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)

Photovoltaic panel emulator in FPGA technology using ANFIS approach

F. Gómez-Castañeda[1], G.M. Tornez-Xavier[1], L.M. Flores-Nava[1],
O. Arellano-Cárdenas[1], J.A. Moreno-Cadenas[1]
[1]Department of Electrical Engineering, CINVESTAV-IPN, Mexico D.F., Mexico
Phone (52) 55 5747 3800 Ext. 6261
E-mail: {fgomez, gtornez, lmflores, arellano, jmoreno} @cinvestav.mx

Abstract — **In this manuscript we present the implementation in FPGA of ANFIS system (Adaptive Network-based Fuzzy Inference Systems) for a two-input architecture with three membership functions per input and nine fuzzy rules, used to set up a photovoltaic panel emulator. The starting point is the photovoltaic panel electric analog model simulated with ELDO, a tool of Mentor Graphics Suite, having as inputs irradiation and temperature from a meteorological data base so we can obtain the short-circuit current (I_{SC}) and open circuit voltage (V_{OC}) of the panel. With this information, ANFIS was trained within Matlab environment to approximate the photovoltaic panel response. The training was carried out for both, current and voltage, independently, and once achieved minimum error parameters, they were downloaded into the FPGA implemented architecture in order to assess its performance.**

Keywords — Photovoltaic panel, ANFIS, neurofuzzy systems, VHDL, FPGA.

I. INTRODUCTION

The use of alternative energy sources has an increasing role to substitute traditional sources; particularly, solar energy results very attractive since it presents an ample availability all around the world and is a clean energy source. Hence, the need of working with photovoltaic systems is increasing day after day. In this work we focus solely in developing the photovoltaic panel model, which is where solar energy is converted to electrical energy. Attaining the analytical model of a photovoltaic panel becomes a complex task, because there are many environmental factors intervening; so, in order to emulate its behavior, we can draw upon artificial neural networks (ANN) as we reported in a previous work [1]. However, with ANNs we cannot get an easily interpretable system and, if there are changes in the data base, the whole system must be trained from scratch and there is no possibility of using prior knowledge.

A neurofuzzy system combines the advantages of fuzzy systems, which deal with the knowledge that can be explained and understood, and ANN that deal with implicit knowledge that can be acquired by means of learning. The learning of a neural network provides a good methodology of adjusting the expert's knowledge (prior knowledge), defined by the fuzzy system, by generating fuzzy rules and adding membership functions automatically, or modifying those that are already defined. Alternatively, the fuzzy logic enhances the capacity of generalization of a neural network, giving it a more reliable output when is required an extrapolation that goes beyond the limits of the training data.

Many developments of neurofuzzy systems have been made in software, taking advantage of the modern computers capacities; however, an important characteristic of these systems is its parallel architecture, which cannot be built precisely with a system that executes programs in a sequential form (software). For this reason, arises the need of using circuits that truly carry out parallel processing, obtaining in this way a better performance, and as the system does not depend on a computer, it might be implemented as a portable system and consequently with less power consumption. This parallelism can be accomplished with devices as FPGAs (Field Programmable Gate Arrays).

The model of the photovoltaic panel to be implemented in the FPGA was approximated with ANFIS and consists of two stages: software implementation, which offers great flexibility for training and simulation of the architecture for general purpose applications, and hardware implementation, that allows to perform real time simulations. The software stage was carried out in Matlab, for training and simulation of the system. In the second stage, timed simulation and FPGA implementation were done by using the Xilinx Suite. The FPGA chip used is one of Spartan 6 family.

II. ANFIS ARCHITECTURE

ANFIS is the straight result of a computation methodology that arose in the 90's called *Soft Computing*, which incorporates the ability of human mind for reasoning and learning in an ambiguous and imprecise environment. ANFIS is a kind of adaptive network that, functionally, is equivalent to a fuzzy inference system [2]. This architecture allows representing directly Sugeno and Tsukamoto fuzzy models. Fig.1 shows the layers that constitute it, in this case for a two-input architecture, with three membership functions for each input and nine fuzzy rules. The five layers that integrate it are the following: Layer 1, membership functions (bell-shape functions with three parameters a_i, b_i and c_i); layer 2, T-norm operator (MIN operator); layer 3, normalization; layer 4, first order polynomials multiplication (with three parameters p_j, q_j and r_j); layer 5, total sum, where i=total number of membership functions and j=total number of fuzzy rules.

In a fuzzy inference system (FIS), the antecedent of a rule defines a local fuzzy region while the consequent describes the behavior inside that region by means of several components. For a Sugeno model, such components can be a constant (order '0' model) or a lineal equation (1st order model). For this last case, a fuzzy rule is described as follows:

978-1-4799-6230-3/14/$31.00 ©2014 IEEE

If x is A_1 and y is B_1, then $z_1 = p_1 x + q_1 y + r_1$

where A_1 and B_1 are linguistic labels (such as “small”, “tall” or “new”).

ANFIS architecture is shown in Fig. 1 The layers drawn with square nodes are adaptable, that is, their values are adjusted when the system is trained; layers drawn with circle nodes remain invariable before, during and after the training. The training is executed in two steps: one forward pass, maintaining the premise parameter (membership functions) fixed, applying the least-squares estimator and having as signals those generated by the node outputs. The other step is a backward pass, where the error signals propagate backward and the premise parameters are updated by gradient descent while holding the consequent parameters (output polynomials) fixed.

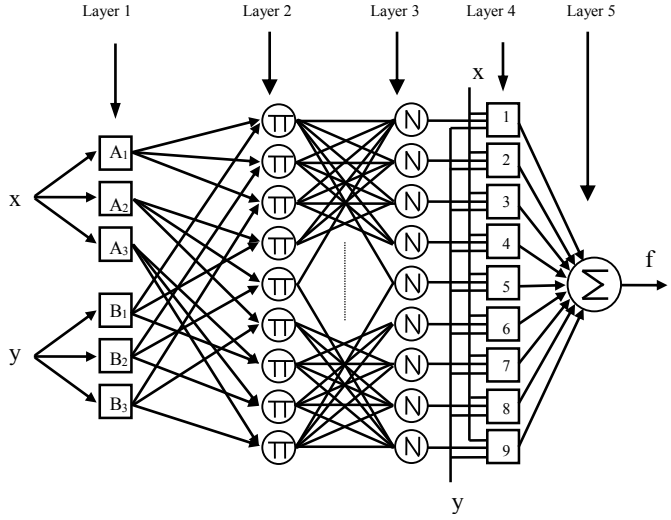

Fig. 1 ANFIS system with two inputs and nine rules.

III. SOFTWARE IMPLEMENTATION

The data set used for training ANFIS should be defined such that it is contained within the two-dimensional fuzzy space established as the universe of discourse for the two input variables of the system (X and Y). For our application, the training data are generated from the known meteorological data base, first performing a normalization so both inputs (radiation and temperature) and output targets (voltage and current) be within the fuzzy space.

Training for current and voltage were done separately to get the sets of premise and consequent parameters to be programed in the FPGA. The total number of vectors in the data base is 1097 that stands for three years of monitoring in the zone, and they were divided as follows: 70% for training (967), 15% for testing (165) and 15% for checking (165).

Fig. 2 shows all the input/output vectors set without normalization, as they were generated from the electric analog model.

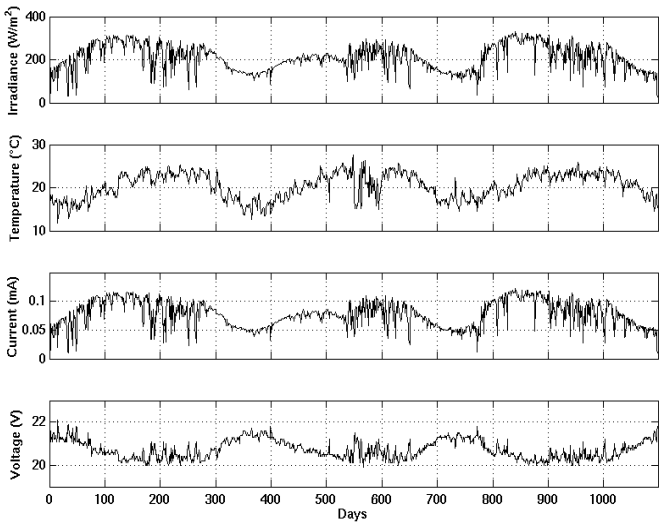

Fig. 2 Inputs (radiation and temperature) and output targets (voltage and current)

Fig. 3 shows Matlab’s ANFIS editor displaying the test vectors for current after 150 epochs of training.

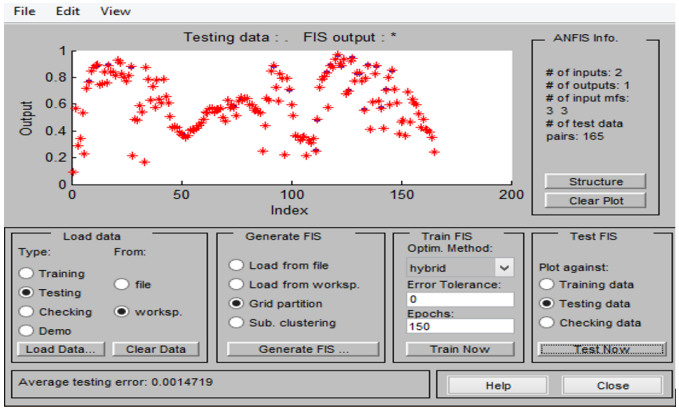

Fig. 3 ANFIS user interface.

After 150 epochs of training, the premise parameters obtained for current are the following:

Input 1:

Bell 1: a = 0.3205, b = 1.992, c = 0.0255
Bell 2: a = 0.2857, b = 2.007, c = 0.511
Bell 3: a = 0.1046, b = 2.008, c = 1.058

Input 2:

Bell 1: a = 0.3245, b = 2.007, c = 0.1798
Bell 2: a = 0.1953, b = 2.009, c = 0.4012
Bell 3: a = 0.184, b = 2.01, c = 0.992

Polynomials [p_j, q_j, r_j]:

1) [1.006 -0.005712 0.0007382]
2) [0.9895 0.007186 -0.0002577]
3) [1.009 0.008716 -0.007701]
4) [0.995 0.005629 -0.001885]
5) [1.009 -0.0004158 -0.005918]
6) [1.005 0.01646 -0.0175]
7) [0.9346 -0.01131 0.05408]
8) [0.9967 0.01971 0.003931]
9) [1.003 0.09357 -0.06962]

The correspondent membership functions for current are shown in Fig. 4.

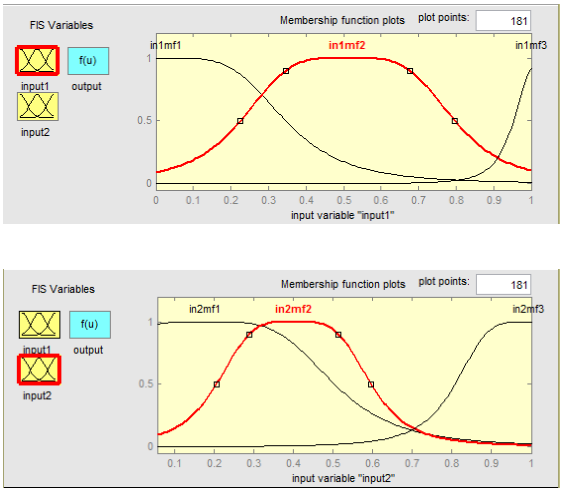

Fig. 4 Membership functions for current.

IV. HARDWARE IMPLEMENTATION

Reviewing the state of the art with regards to ANFIS implementation in hardware, we reviewed three papers [3, 4, 5] where they perform serial processing and, in this way, they consume less logical resources of the FPGA. In contrast, we implement parallel processing in order to achieve the smallest response time of the system.

In order to test ANFIS architecture, we elaborated the description of the system in VHDL language in Xilinx's development tool ISE version 14.2, and created all the necessary functions for each layer. The final system is an ANFIS architecture with two inputs, three membership functions per input and nine fuzzy rules that were implemented in a FPGA device Spartan-6 XC6SL45, and the logical and timed simulations were done in *ModelSim SE 10.0c*.

From the training results (membership functions and polynomials parameters) we decided to use a fixed point representation with signed numbers in two's complement [6], and a word size of 16-bit that was split as follows (Fig. 5).

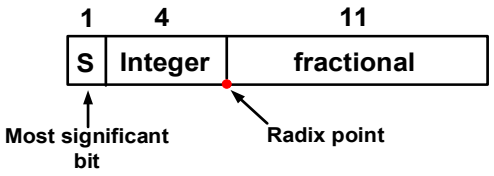

Fig. 5 Binary representation.

In Fig. 6 the blocks diagram of ANFIS as it was implemented in the FPGA is depicted. It is conformed of three main subsystems that are representative of ANFIS architecture: 1) membership functions; 2) fuzzy rules evaluation and MIN operator; 3) defuzzyfier. Control signal (CTRL) has a period of 100 ns and pulse width of 5 ns which is use to enable the memory blocks.

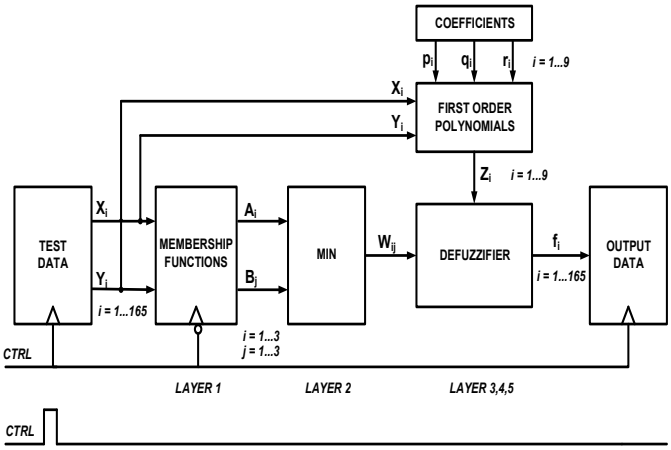

Fig. 6 ANFIS digital system.

a) *Test data.*

This block has two banks of random access memory (RAM), where are stored the test vectors (radiation and temperature) used during the training in Matlab, in this way, each memory bank has 165 16-bit-words. The access to these memories is carried out with the rising edge of CTRL signal.

b) *Membership functions.*

This subsystem computes the grade of membership for each input vector starting with the falling edge of CTRL signal. It consists of two RAM memory banks of 2048 memory locations since the universe of discourse is defined within [0 1] interval. See Fig. 7.

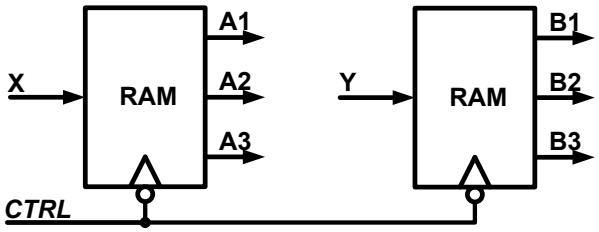

Fig. 7 Memory blocks of membership functions.

Fig. 8 shows the VHDL implementation of membership functions in *ModelSim* for radiation data.

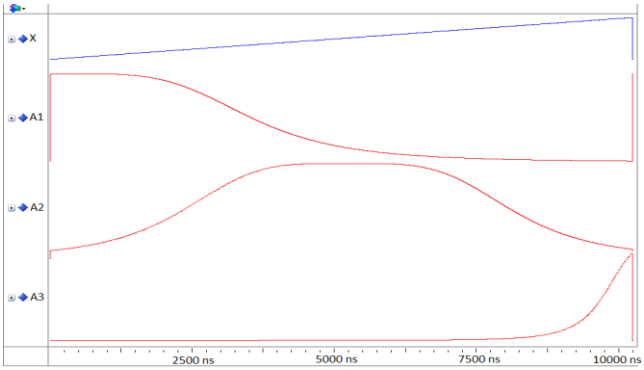

Fig. 8 Current membership functions for input 1.

c) *MIN operator.*

This subsystem evaluates the antecedents of each rule to generate all the possible permutations of them. As we have three membership functions per input, there will be nine permutations (W_{ij}) corresponding to the strength of each rule. The T-norm operator utilized is the MIN, which is implemented through a magnitude comparator that selects the minimum antecedent by means of a multiplexer as shown in Fig. 9.

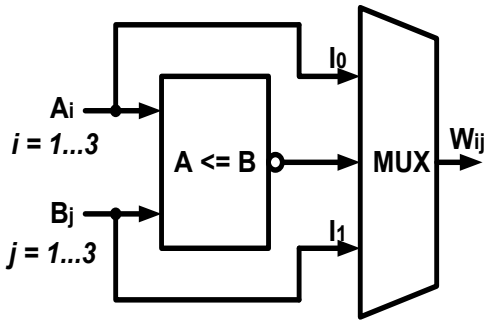

Fig. 9 Fuzzy rules evaluator subsystem.

d) *1st order polynomials.*

The polynomials coefficients [p_i, q_i, r_i] generated by training in Matlab are stored as constants in registers to calculate Z_i, as shown in (1).

$$Z_i = Xp_i + Yq_i + r_i \quad (1)$$

In this way, the system consequents are computed as depicted in Fig. 10.

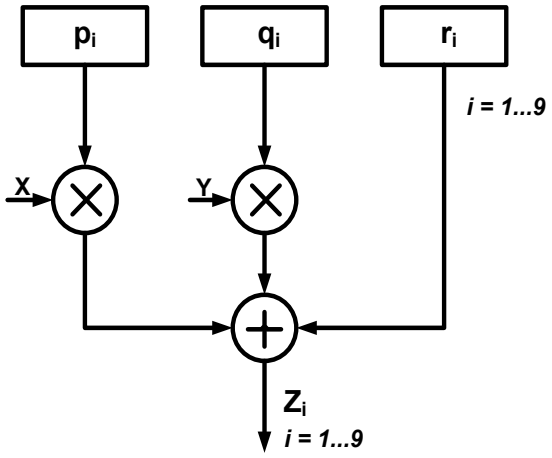

Fig. 10 1st order polynomials evaluation.

e) *Defuzzifier.*

This subsystems delivers the crisp output of ANFIS (f_i) by realizing the operations of layers 3, 4 and 5 of Fig.1, and is estimated as shown in (2).

$$f_i = \frac{\Sigma W_{ij} Z_i}{\Sigma W_{ij}} \quad (2)$$

This subsystem executes 3 tasks:

1. Multiplies the rule strengths from *MIN operator* subsystem by its consequent ($W_{ij} * Z_i$).

2. Calculates the numerator which is the sum of products ($W_{ij} * Z_i$), and the denominator which is the sum of all the rule strengths W_{ij}.

3. Evaluates the quotient of previous numerator and denominator to obtain the crisp output of ANFIS (f_i) (Fig. 11).

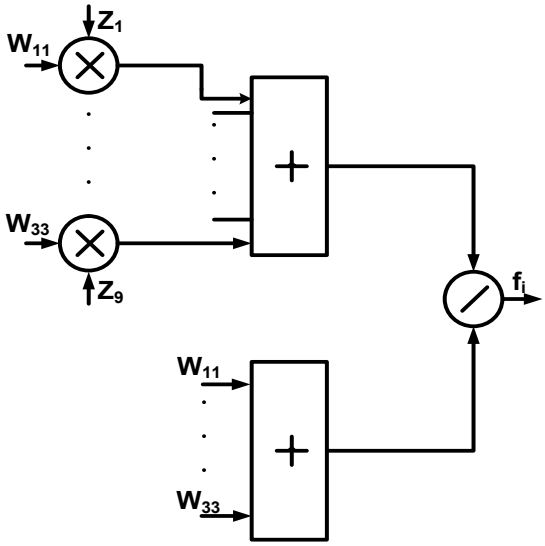

Fig. 11 Defuzzifier

f) *Output data.*

Output data of (f_i) are stored in a RAM block, whose access is in the rising edge of *CTRL* signal. They will be used later for analysis. Fig. 12 shows the simulated output of ANFIS plotted in ModelSim in analog format.

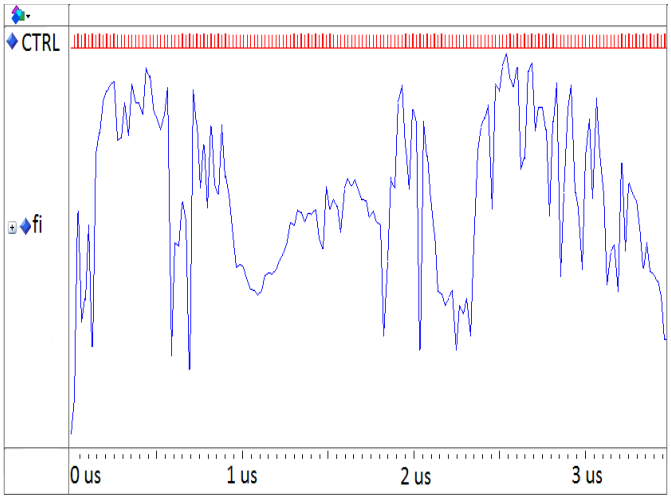

Fig. 12 Output data f_i for the short-circuit current.

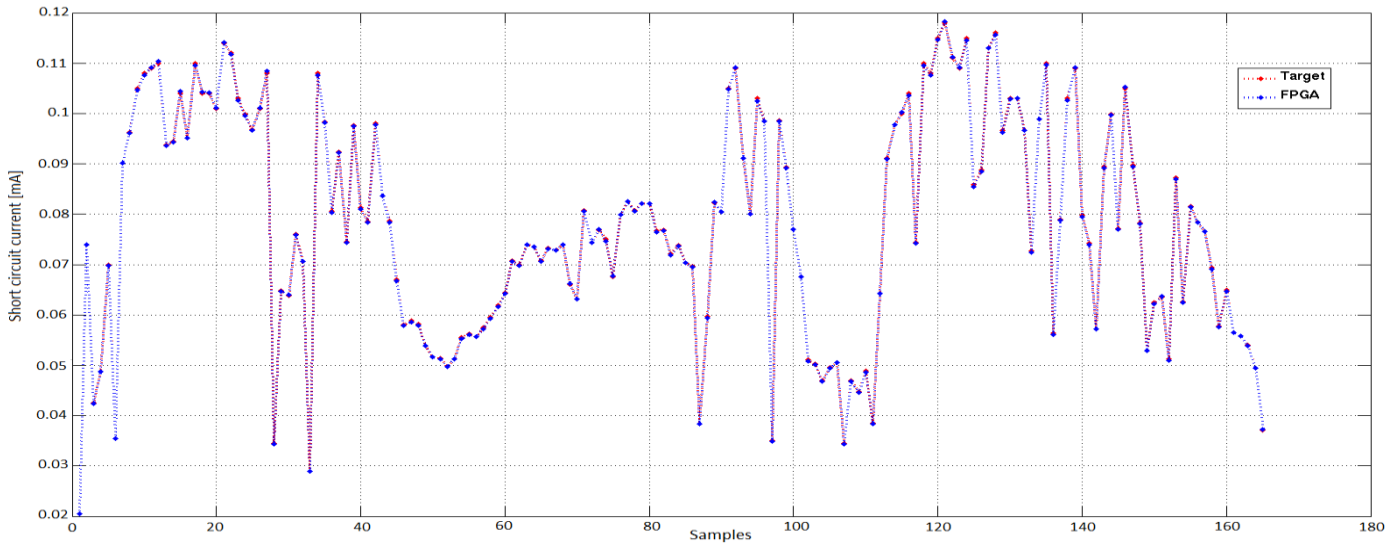

Fig. 13 Comparison between output current of photovoltaic panel and ANFIS implementation on FPGA.

Logical resources utilized.

Table 1 shows the additional logical resources used in the Spartan-6 XC6SL45 device for the ANFIS implementation. As we can see, the resources most used are the multipliers embedded within DSPs blocks due to our parallel process.

Table 1. Resources used for the FPGA

Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	936	27288	3%
Number of Slices	446	6822	6%
Number of RAM B16BWERs	20	116	17%
Number of RAM B8BWERs	2	232	1%
Number of DSP48A1s	31	58	53%

V. RESULTS

In this paper we have emphasized simulations and results for short-circuit current, however the same was done for open circuit voltage. Due to the fact ANFIS has just one output, in the FPGA implementation there is a multiplexer that uses a control signal to interchange the calculated membership functions and polynomials coefficients either for voltage or current, so the system will work for one or the other.

Fig. 13 displays a graphical comparison between short-circuit current from photovoltaic panel (red line) and the output data produced by the FPGA implementation (blue line); as we can observe, visually there is a very high matching that allows us to affirm that the VHDL ANFIS implementation can properly emulate the behavior of the photovoltaic panel.

Below there is a scatter graph showing the regression line and the lineal equation that allows us to predict the output that will have our ANFIS system for any arbitrary target, also it is shown the Pearson correlation coefficient (R=0.9999), which indicates the degree of dependence existing between expected test data (targets) and output data generated by our digital system (Fig. 14).

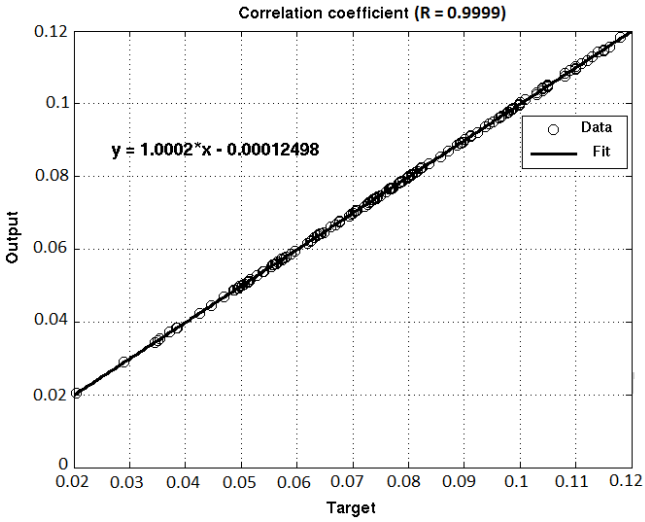

Fig. 14 Regression graph Target vs. Output.

In Fig. 15 is depicted the response time of our digital implementation measured with a Tektronix MSO 2012 oscilloscope, where it is registered a 70ns delay among CTRL fall edge (blue) and ANFIS output signal f_i (red), after all the bits reached its stable value for a given input vector that originated the worst delay.

Table 2 shows a comparison of the results obtained with digital ANFIS and the ANN architecture presented in a previous paper [1]. There are shown the mean square error (MSE), the Pearson correlation coefficient (R) and the response time.

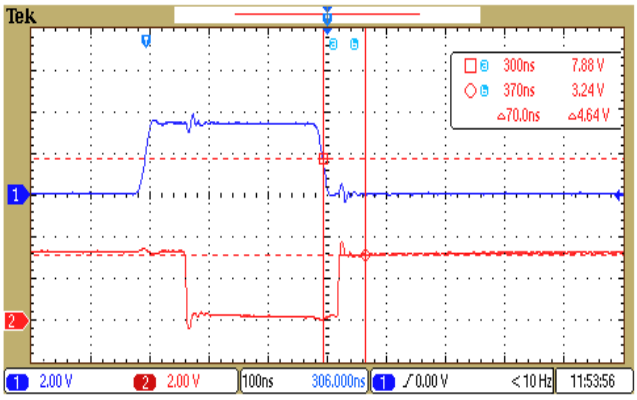

Fig. 15 Response time of digital ANFIS system.

Table 2. Comparison between ANN and ANFIS implementations.

	RNA (7-9-2)		ANFIS	
	I_{SC}	V_{OC}	I_{SC}	V_{OC}
MSE	2.0182E-07	10E-04	3.9939E-08	9.7904E-04
R	0.9998	0.9972	0.9999	0.9973
t_{pd}	1200 ns		75 ns	

VI. CONCLUSIONS

In this work, we implemented an ANFIS architecture for emulating a photovoltaic panel that can operate in real time, this characteristic is very useful for analysis in the lab and in the field. We observed response times from 40 to 75 ns (the worst case) due to our parallel processing.

From results shown in table 2, we note that the photovoltaic panel emulator implemented with ANFIS and that realized with ANN, produce suitable values for short-circuit current (I_{SC}) and open circuit voltage (V_{OC}) so, with any implementation it is possible replicating the behavior that would have a commercial photovoltaic panel when excited with radiation and temperature.

The choice to employ ANN or ANFIS implementation will rely on the application speed needs, as ANFIS response time is significantly less, but with regards to error and utilized resources of the FPGA [1] we cannot conclude which one has the better performance.

REFERENCES

[1] G.M Tornez, F.Gómez, J.A. Moreno, L.M. Flores "FPGA Development and implementation of a solar panel emulator". *In 10th International on Electrical Engineering, Computing Science and Automatic Control CCE 2013* .

[2] J. S. R. Jang, C. T. Sun and E. Mizutani, "Neuro –Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence". Prentice Hall, 1997.

[3] P. Pande, P. Paikrao, D. Chaudhari "Digital ANFIS Model Design". *International Journal of Soft Computing and Engineering (IJSCE), Vol 3, March 2013*.

[4] H. Saldaña, C. Cárdenas "Design and implementation of an adaptive neuro fuzzy inference system on an FPGA used for nonlinear function generation". *IEEE Third Latin American Symposium on Circuits and Systems (LASCAS), pp.1-5, 2010*.

[5] H. Saldaña, C. Cárdenas "A digital hardware architecture for a three-input one-output zero-order ANFIS". *IEEE Third Latin American Symposium on Circuits and Systems (LASCAS), pp.1-4, 2012*.

[6] A. Savich, M. Moussa, S. Areibi "The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study". *IEEE Trans. on Neural Networks, Vol.18, No.1, January 2007*.