Efficient Hardware/Software Implementation of an Adaptive Neuro-Fuzzy System

Inés del Campo, *Associate Member, IEEE,* Javier Echanobe, *Member, IEEE,* Guillermo Bosque, and José M. Tarela

***Abstract*— This paper describes the development of efficient hardware/software (HW/SW) neuro-fuzzy systems. The model used in this work consists of an adaptive neuro-fuzzy inference system (ANFIS) modified for efficient HW/SW implementation. The design of two different on-chip approaches are presented: a high-performance parallel architecture for off-line training, and a pipelined architecture suitable for on-line parameter adaptation. Details of important aspects concerning the design of HW/SW solutions are given. The proposed architectures have been implemented using a system-on-a-programmable-chip (SOPC). The device contains an embedded-processor core and a large field programmable gate array (FPGA). The processor provides flexibility and high precision to implement the learning algorithms, while the FPGA allows the development of high-speed inference architectures for real-time embedded applications.**

***Index Terms*— Adaptive systems, embedded systems, field programmable gate array (FPGA), neuro-fuzzy model, system-on-a-programmable-chip (SOPC)**

I. Introduction

HYBRID neuro-fuzzy systems (NFS) combine artificial neural networks and fuzzy logic in a synergetic way. Fuzzy systems provide a framework to represent imprecise information and to reason with this kind of information, while neural networks enhance fuzzy systems with the capability of learning from input-output samples; learning is used to adapt parameters of the fuzzy system as membership functions or rules. In the last decade NFSs have become very popular mainly due to the powerful capabilities as universal function approximators that most of them exhibit, even when simple membership functions like trapezes or triangles are used [1]–[8]. Some example representative application areas of NFSs are: automatic control, robotics, adaptive signal processing, pattern recognition, and system identification (see, for example, [9]). More recent applications of NFS can be found in [10]-[17]. This paper deals with efficient implementations of a class of NFS, the adaptive neuro-fuzzy inference system (ANFIS) [18], [19], that has been widely used to develop NFSs in the above application areas. ANFIS is a network representation of different types of fuzzy inference models, endowed with the learning capabilities of neural networks. In concrete, our work focuses on an ANFIS-like model that is functionally equivalent to the Takagi-Sugeno inference system [20], [21].

In the last two decades, several researchers have developed special purpose hardware in the fields of fuzzy systems, neural networks, and their combinations (see, for example the special issues [22]-[25], and references therein). Restrictive design specifications such as high performance, reduced size, or low power consumption, which are difficult to fulfil with a software approach, have been the main reasons for developing these solutions. However, both the research activity and the commercial interest in neural/fuzzy hardware have been decreasing in the last years due to the important increase in speed of software solutions based on general-purpose microprocessors or digital signal processors (DSP). Nowadays, software approaches traditionally characterized by their high versatility, also provide processing speeds that are high enough to develop a large number of neural/fuzzy applications. In this context, dedicated hardware implementations provide a suitable solution only when extreme requirements –in terms of speed, power consumption, or size– are needed.

On the other hand, in recent years heterogeneous hardware/software (HW/SW) technologies have emerged as an optimal solution for many systems where a trade-off between versatility and performance is required. This approach proposes the partition of the system into hardware (HW) and software (SW) parts by exploiting the advantages of both HW and SW intrinsic characteristics [26], [27]. In [28], a comparative analysis of different approaches to the implementation of NFSs suggests that HW/SW solutions can often outperform homogeneous solutions based either on HW or SW. In particular, to take full advantage of HW/SW solutions, it would be desirable for all the parts of the adaptive NFS to be integrated in a single chip. In this sense, new Field Programmable Gate Arrays (FPGAs) are powerful enough to accommodate all the components of a typical embedded

Manuscript received November 3, 2006; revised March 12, 2007. This work was supported in part by the University of the Basque Country under Grant UPV224.310-E-15871/2004, and the Basque Country Government under Grant SA-2006/00015.

I. del Campo, J. Echanobe, and J. M. Tarela are with the Department of Electricity and Electronics, University of the Basque Country, Leioa, Vizcaya, 48940 Spain (e-mail: ines@we.lc.ehu.es).

G. Bosque is with the Department of Electronics and Telecommunications, University of the Basque Country, Bilbao, 48012 Spain.

system (e.g. processor cores, memory blocks, peripherals, specific HW, etc) on a single chip, commonly referred to as System-on-a-programmable-chip (SOPC). Therefore, HW/SW implementations can also benefit from the well known advantages of FPGA technology such as short time-to-market, high flexibility, re-usability, and availability of IP (intellectual property) cores, among others.

This work presents the development of an adaptive NFS, modified for efficient HW/SW implementations. The system has been successfully implemented using a SOPC of Altera´s Excalibur family [29]. The device features an ARM embedded-processor core together with a large FPGA for application-specific HW. Two different on-chip HW/SW architectures have been developed. The first one consists of a high-speed parallel architecture with fixed parameters (off-line learning), while the second one is a pipelined architecture suitable for on-line learning. The proposed approach is well suited for the development of real-time embedded NFSs. In particular, as will be seen, the efficiency of our proposals increases when dealing with systems with a large number of inputs.

The paper is organized as follows: Section II briefly overviews the generic ANFIS architecture and introduces the particular class developed in this work, the piecewise multilinear (PWM) ANFIS. In Section III the approximation capability and the learning performance of the proposed model are analyzed. Section IV addresses some practical design considerations concerning the development of efficient implementations such as the partition of the system into HW and SW blocks, the selection of a digital word-length, and the problem of HW/SW communication. Section V presents the developed architectures and gives details of their SOPC-based implementations. Special attention is paid to the analysis of the efficiency of the proposed solutions. Finally, Section VI summarizes with some concluding remarks.

II. Adaptive Neuro-Fuzzy Model

The NFS system developed in this work is an ANFIS-like model modified for efficient HW/SW implementations. First, for better understanding of the advantages of the proposed modifications or constraints, let us introduce the basics of the ANFIS model [18], [19], for the case of a zero-order Takagi-Sugeno inference system [20], [21]. Consider a set of fuzzy rules:

R_j: IF x_1 is $A_{1j}(x_1)$ and x_2 is $A_{2j}(x_2)$ and . . . x_n is $A_{nj}(x_n)$ THEN y is c_j,

where R_j is the jth rule ($1 \leq j \leq m$), x_i ($1 \leq i \leq n$) are input variables, y is the output, c_j is a constant consequent, and $A_{ij}(x_i)$ are linguistic labels each one being associated with a membership function $\mu_{ij}(x_i)$. In a zero-order Takagi-Sugeno fuzzy model the inference procedure used to derive the conclusion for a specific input $\mathbf{x} = (x_1^0, x_2^0, \ldots, x_n^0)$ consists of two main steps. First the firing strength or weight w_j of each rule is calculated as

$$w_j = \prod_{i=1}^{n} \mu_{ij}(x_i^0) . \tag{1}$$

After that, the overall inference result, y, is obtained by means of the weighted average of the consequents

$$y = (\sum_{j=1}^{m} w_j c_j) / (\sum_{j=1}^{m} w_j) . \tag{2}$$

Equations (1) and (2) provide a compact representation of the inference model.

ANFIS consists in a representation of different types of fuzzy inference models as adaptive networks. In concrete, the above fuzzy model can be viewed as an adaptive network with the following layers (see Fig. 1):

Layer 1 is composed of m groups of n nodes each one. Every node (i,j) in this layer is an adaptive node that produces output $O_{ij}^{(1)}$ by evaluating the corresponding membership function

$$O_{ij}^{(1)} = \mu_{ij}(x_i^0) = f(x_i^0; p_{1_{ij}}, p_{2_{ij}}, \cdots), \; 1 \leq i \leq n \text{ and } 1 \leq j \leq m, \tag{3}$$

where $(p_{1_{ij}}, p_{2_{ij}}, \cdots)$ are the parameters associated with each antecedent membership function (e.g. centre, and width); by changing the value of these parameters the membership functions can be adjusted.

Layer 2 contains m nodes with outputs $O_j^{(2)}$. Node j in this layer generates the firing strength of the j-th rule by computing the algebraic product of all its inputs,

$$O_j^{(2)} = \prod_{i=1}^{n} O_{ij}^{(1)} = w_j , \text{ with } 1 \leq j \leq m. \tag{4}$$

Layer 3 is an m-nodes normalization layer. This layer performs the normalization of the activation of the rules; the output of the j-th node, $O_j^{(3)}$, is the ratio of the j-th rule's weight to the sum of the weights of all the rules:

$$O_j^{(3)} = w_j / (\sum_{j=1}^{m} w_j) . \tag{5}$$

Layer 4 contains only one node. The node output, $O^{(4)}$, is the weighted sum of the consequents,

$$O^{(4)} = \sum_{j=1}^{m} O_j^{(3)} c_j = y \tag{6}$$

this node is an adaptive node whose parameters, c_j, are the set of consequent parameters.

The layered structure (3) to (6), viewed as a neural network, can adapt its antecedent and consequent parameters to improve the system performance or to deal with dynamic changes.

A. The PWM ANFIS Model

In order to reduce the complexity of the above ANFIS model, let us introduce the following restrictions on the antecedents: (i) the membership functions are overlapped by pairs, (ii) they are triangular shaped, and (iii) they are normalized in each input dimension. Similar constraints have been successfully used by many designers to simplify digital and analogue approaches of fuzzy HW and adaptive neuro-fuzzy HW [30]-[33]. In the following we will analyze the advantages of constraints (i) to (iii) on the simplicity of the layered representation of the fuzzy system and also on the parameter adaptation procedure. First, let us consider some immediate consequences of these restrictions. The first restriction forces the overlapping degree of the antecedents to be two. Therefore, given an input vector $\mathbf{x}=(x_1^0,x_2^0,\ldots,x_n^0)$, only two antecedents per input dimension provide membership values different from zero (i.e. active antecedents). To be exact, due to (ii) and (iii), only one half of the triangles concerned becomes active. Fig. 2 depicts typical membership functions for a two-input system, with 4 triangular antecedents per input, verifying the above constraints. It can also be seen in this figure that the vertex of the 4 triangles delimits 3 intervals per axis that induce a total of 9 rectangular cells in the two-dimension input space. As a general rule, if NA_i is the number of triangles of the i-th input, $1\leq i\leq n$, then, the overlapping of these triangles delimits NA_i-1 intervals per input, r_i, which induces a partition of the input space into $\prod_{i=1}^{n}(NA_i-1)$ polyhedral cells or regions. Our interest in this geometrical view of the input domain is due to the fact that only one of these cells is involved in the calculus of the system output at each time. The whole system can, therefore, be implemented as a single inference kernel [30], [34]-[36]. The parameters of the kernel depend on the concrete region where the input vector falls (i.e. active region). As will be seen below, the modified ANFIS generates a piecewise multilinear (PWM) output. For this reason, in the rest of the paper we will refer to this system as the PWM ANFIS. This network can be organized into the following active layers (see Fig. 3):

Layer 1. Every node in this layer computes one active triangular-shaped membership function. In virtue of (i), each input is concerned with two membership functions per dimension. Therefore, we have only 2n active nodes in this layer, independent of the number of antecedents per input dimension. In addition, due to (iii), each pair of active nodes ($O_i^{(1)},\bar{O}_i^{(1)}$) are complementary, that is $O_i^{(1)}+\bar{O}_i^{(1)}=1$, where the upper bar means fuzzy complement (see Fig. 4). The node outputs are as follows:

$$\begin{cases} O_i^{(1)}=\mu_i^{r_i^0}(x_i^0)=a_i^{r_i^0}(x_i^0-b_i^{r_i^0}) \\ \bar{O}_i^{(1)}=1-\mu_i^{r_i^0}(x_i^0) \end{cases},\ 1\leq i\leq \text{n}, \tag{7}$$

the super-index r_i^0 denotes the active interval of the i-th input, $a_i^{r_i^0}$ denotes the positive slope of the active semi-triangles, and $b_i^{r_i^0}$ is the offset of the interval with respect to the origin.

Layer 2. Each node in this layer generates a multilinear output which represents the firing strength of a rule; each multilinear term consists in the product of n linear terms like (7). The rules with non-zero firing strength are only those associated with the active neurons in layer 2 (only one pair of complementary neurons per input), that is, 2^n active rules. If $\mathbf{j}$ denotes the neuron index (j) of layer 2, but codified as an n-bit binary word, $\mathbf{j}=(j_n j_{n-1}\cdots j_1)$, then,

$$O_j^{(2)}=w_j=\prod_{i=1}^{n}\phi_{j_i},\ \text{with}\ \phi_{j_i}=\begin{cases} O_i^{(1)} & \text{if}\ j_i=1 \\ \bar{O}_i^{(1)} & \text{if}\ j_i=0 \end{cases},\ 0\leq j\leq 2^n\text{-}1. \tag{8}$$

The notation that we propose above is useful for HW description purposes because the bits of the neuron index can be directly used to construct the set of active rules by selecting all possible combinations of neurons (`1′) and their complements (`0′).

Layer 3. Taking into account constraint (iii), it can easily be proved [4] that $\sum_{j=0}^{2^n-1}w_j=1$, therefore the normalization layer (5) disappears because the divide operation is unnecessary. Finally, the output layer is reduced to the sum of 2^n product terms,

$$O^{(3)}=\sum_{j=0}^{2^n-1}O_j^{(2)}c_j^0=\sum_{j=0}^{2^n-1}w_j c_j^0=y \tag{9}$$

where c_j^0 denotes the active consequents. It has been seen that the main benefits of constraints (i) to (iii) on the general ANFIS architecture are a reduction of the number of neurons per layer (layers 1 and 2) due to the activation of a reduced number of antecedents, the elimination of the normalization layer, and a simplification of the network arithmetic.

B. Learning Algorithm

As has been seen, the restrictions imposed on the antecedent membership functions have a great impact on network simplicity. In the same way, it will be shown how the learning procedure is also significantly simplified. Although a basic back-propagation learning rule can be used to adapt the set of network parameters, the learning process generally becomes too slow. To avoid this problem, the hybrid learning rule proposed in [18], which combines the gradient-descent method (GDM) and the least-square estimator (LSE), is used. Each epoch of the hybrid procedure is composed of a forward pass and a backward pass. In the forward pass the consequent parameters are identified by the LSE method and in the backward pass the antecedent parameters are updated by the GDM.

The advantages of restrictions (i) to (iii) concerning the learning procedure are twofold. Firstly, as has been seen, the PWM ANFIS limits the activation of the system each time to a single cell or region of the input space. This cellular nature of the feed-forward network (fuzzy inferences) also reduces the computational complexity of the learning algorithm because both LSE and GDM equally require the evaluation of the feed-forward network. Secondly, the constraints imposed on the input partition reduce not only the active set of parameters for a concrete input (parameters of the active cell) but also the total set of antecedent parameters. That is, (see Fig. 4) the partition is completely defined by giving the triangle offsets, $b_i^{r_i}$; the triangle slopes are obtained as $a_i^{r_i}=1/(b_i^{r_i+1}-b_i^{r_i})$. Moreover, since the position of the first and the last pairs of triangles is determined by the bounds of the input space, the total number of antecedent parameters to be trained in a PWM ANFIS with NA_i antecedents per input is equal to NA_i-2, $1 \leq i \leq n$. For example, a single input system with a partition of the input space like that depicted in Fig. 4 (NA_i=5), requires the adaptation of only 3 antecedent parameters against the 10 parameters required to define symmetrical functions like triangles or Gaussians (5 centres and 5 widths), or even the 15 parameters required to define unconstrained asymmetrical triangles. Concerning the consequent parameters, the number of crisp consequents to be adapted is equal to the number of rules, $m = \prod_{i=1}^{n} NA_i$, where all possible rules are considered in order to guarantee the completeness of the rule set.

Let us briefly present both -the LSE and the GDM- as applied to our system. Note that although only 2^n rules become active for each training data, the whole set of rules is involved in the training procedure. Hence equation (9) can be rewritten as

$$y = w_1(\mathbf{x})c_1 + w_2(\mathbf{x})c_2 + \cdots + w_m(\mathbf{x})c_m \tag{10}$$

where $\mathbf{x}$ is the vector of input variables. Since y is linear in the consequent parameter, c_j, given values of the antecedent parameters, and a set of training data pair $\{(\mathbf{x}_k; y_k^{'}), k = 1,\ldots,K\}$, each training pattern can be substituted into (10) to obtain a set of K linear equations. Equation (10) represents a typical LSE problem that can be solved directly or recursively [37].

After the consequent parameters have been identified, the GDM is used to optimize the shape of the NA_i triangles in the partition of the i-th input. Consider the error function

$$E = \frac{1}{2K}\sum_{k=1}^{K}(y_k - y_k^{'})^2 \tag{11}$$

where y_k is the actual output of the network for the k-th training data and $y_k^{'}$ is the desired output. In the off-line operation mode the parameter update is performed after the evaluation of all the training patterns. The learning rule for the triangle offsets, $b_i^{r_i}$, with $1 \leq r_i \leq NA_i-2$, is

$$b_i^{r_i(t+1)} = b_i^{r_i(t)} - \eta_{b_i}\left(\frac{\partial E}{\partial b_i^{r_i}}\right) \tag{12}$$

where η_{b_i} is the learning rate for the offsets of the i-th input. The chain rule has been used to calculate the above derivative (see Appendix)

$$\frac{\partial E}{\partial b_i^{r_i}} = \frac{1}{K}\sum_{k=1}^{K}(y_k - y_k^{'})\sum_{j=0}^{2^n-1} c_j^0 w_j f_{j_i}^{r_i^0} \tag{13}$$

with $f_{j_i}^{r_i^0} = -1/(x_i^0 - b_i^{r_i^0}) + 1/(b_i^{r_i^0+1} - b_i^{r_i^0})$ if j_i=1, and $f_{j_i}^{r_i^0} = 1/(b_i^{r_i^0+1} - b_i^{r_i^0})$ if j_i=0, where j_i has been defined in (8).

The learning rule (12)-(13) consists in simple arithmetic operations, subtractions and sum of products that can be efficiently implemented as SW using a microprocessor or a DSP. Note that, in addition to the aforementioned advantages, in our system only one kind of parameter is to be trained in the backward stage, namely, the triangle offsets, while in the general case at least two different types of parameters are involved (e.g. centre and width). The above hybrid learning rule is suitable for off-line learning and also for on-line learning with minor modifications [37].

III. MODEL VALIDATION AND SIMULATION RESULTS

Before introducing the HW/SW implementation of the PWM ANFIS, as we are dealing with a constrained model, the problem of its actual modeling capabilities needs to be investigated. It is important to analyze both the approximation capability of the model and its learning performance when the hybrid learning algorithm is used. To perform computer simulations the authors developed a set of flexible Matlab m-functions for multidimensional PWM ANFIS (the PWM-ANFIS Toolbox). The Toolbox includes also fixed-point functions to simulate finite precision computation.

A. Approximation Capabilities of the PWM ANFIS

The approximation capability of the PWM ANFIS is that of the zero-order Takagi-Sugeno inference model, verifying the restrictions (i) to (iii). This model has been analyzed in [4] where the author shows that the system is able to approximate, to any degree of accuracy, not only sufficiently regular functions, but also their derivatives, while keeping the linguistic interpretability of their fuzzy rules. In other words, the restrictions imposed on the model with the aim of benefiting efficient implementations do not hinder its approximation capability. Moreover, as has been pointed out in the previous Section, a consequence of the restrictions is a reduction of the system complexity; in particular, the number of neurons of the network does not depend on the number of membership functions per input dimension. This means that any input-output relationship can be matched well arbitrarily by refining the partitions of the input universes, without sacrificing the simplicity of the implementations.

B. Learning Ability

Concerning the learning abilities of the system, we have examined the learning performance of our system using several test functions. In each experiment, we have compared the performance of the PWM ANFIS with that of the unrestricted ANFIS with Gaussian antecedents. We have also evaluated our results in comparison with other meaningful approaches for NFS modeling [38]-[40]. The experiments start with the specification of an initial input partition using symmetrical membership functions for the antecedents. In each iteration, the singleton consequents are identified by means of the LSE method and then the antecedent parameters are updated by using the GDM. Two nonlinear functions commonly used to test learning approaches have been selected as representative case examples from a more comprehensive study performed by the authors. The object of our experiments is not to improve the results obtained by other methods, but to investigate if the learning performance of the PWM ANFIS is satisfactory. The experiments are evaluated in terms of the mean squared error (MSE) between the target functions and their corresponding approximations.

1) Experiment 1: The first experiment considers the modeling of the nonlinear function

$$y = x_1 \sin(x_2) + x_2 \cos(x_1),\ 0 \le x_1, x_2 \le \pi. \quad (14)$$

A graphical representation of the function is shown in Fig. 5(a). The training data pairs $\{(\mathbf{x}_k; y'_k),\ k = 1,\ldots,K\}$, have been generated by sampling the input universes with a sampling interval of $\pi/20$. As a result, the number of training samples is 441 (K=441). The PWM ANFIS used in this experiment involves four membership functions per input variable (i.e., 16 rules). Therefore, the total number of parameters to be adapted is 20, including 4 antecedent parameters (2 offsets per input) and 16 consequent parameters. The Gaussian ANFIS trained for comparison involves the same number of antecedents and rules, but it requires the adaptation of a larger number of parameters, 32 to be exact; 16 antecedent parameters plus 16 consequent parameters. The evolution of the MSE during the learning process for both our system and the Gaussian ANFIS is shown in Fig. 6. As can be seen, the main consequence of the restrictions imposed on our system is a greater MSE, but of the same order of magnitude as for the Gaussian ANFIS during the 200 iterations of our study. For example, iteration 4 gives MSE=0.0016 for the Gaussian ANFIS, and MSE=0.0060 for our system. Iteration 6 gives MSE=0.0012 and MSE=0.0047, and iteration 8 produces MSE=0.0010 and MSE=0.0044 respectively. As evidenced in Fig. 6, our system does not achieve significant error reduction after iteration 8, while the Gaussian ANFIS does continue improving the approximation. This behaviour is not surprising taking into account that the flexibility of smooth Gaussian membership functions is greater than that of restricted triangular-shaped membership functions [41]. The approximated function that our system provides from the eighth iteration is shown in Fig. 5(b).

However, the most important feature required for an adaptive network is its generalization ability, that is, the ability of the system to provide satisfactory results when it is evaluated using a collection of non-training data. The computation of the MSE in these kind of data is usually referred to as the generalized mean squared error (GMSE). Table I presents a comparison of the generalization ability of our system and meaningful earlier works, Lee's system [38], Wong's system [40], and Lin's system [39]; the results were calculated in [38]. Lin's system uses trapezoidal membership functions, while Wong's and Lee's systems use Gaussian membership functions. All of them implement a structure identification step to construct an initial fuzzy model. After that Lin and Wong use GDM for parameter identification, while Lee uses a hybrid learning algorithm (LSE and GDM). As can be seen in Table I, the GMSE of our system is comparable to those provided by the other methods. Taking into account that the works selected for comparison start the learning procedure from better initial models, we consider this a meaningful result. Finally, we investigated the GMSE reduction by refining the partition of the input universes of our system. As was expected, these experiments report better approximations. For example, with 5 antecedents per input the PWM ANFIS gives GMSE=0.0015 and with 6 antecedents per input our system gives GMSE=0.0007. The surface obtained by means of 6 antecedents per input has been represented in Fig. 5(c).

2) Experiment 2: The second example concerns the modeling of the following nonlinear function

$$y = (1 + x_1^{-2} + x_2^{-1.5})^2,\ 1 \le x_1, x_2 \le 5. \quad (15)$$

A set of 50 input-output patterns has been used to train the network. The input universes have been partitioned into only 3 membership functions each, giving as a result 9 fuzzy rules. The trainable parameters are 2 offsets (1 per input) and 9 consequents for our system, while the Gaussian ANFIS involves 6 antecedent parameters and 9 consequents. Fig. 7(a) shows the target function and Fig. 7(b) shows the PWM ANFIS approximation after the 25-th iteration. The MSE in this iteration is of 0.0043 while the MSE for the same iteration but using ANFIS with Gaussian membership functions is of 0.1217. As can be seen in Fig. 8, our system approximates better than the Gaussian ANFIS up to iteration 60 approximately, where both systems give similar errors. After that, Gaussian functions continue evolving and improving the approximation (for example, iteration 200 gives MSE=0.0004 for the Gaussian ANFIS).

With respect to the generalization ability of the system, the comparisons with other authors using a set of non-training data are given in Table I where the GMSE errors are listed. The systems considered for comparison are the same as for experiment 1. As can be seen, our PWM ANFIS approximates the target function at test points almost as well as the other systems. Finally, we observed that in this experiment a finer partition of the inputs, with a larger number of membership functions, does not reduce significantly the GMSE due to the reduced set of training data. If better precision is required, a greater number of training patterns needs to be used along with the partition refinement.

Our results lead us to conclude that the learning performance of PWM ANFIS is satisfactory and comparable to those reported by other authors using more elaborate schemes. The PWM ANFIS provides fast convergence due to the use of a hybrid learning algorithm and due to the reduced number of adjustable parameters. This is specially interesting in the implementation of applications that require on-line parameter adaptation.

IV. HW/SW DESIGN CONSIDERATION

Next we will focus on some important design aspects concerning the HW/SW implementation of the PWM ANFIS on a SOPC. The device used in this work is a SOPC of Altera´s Excalibur family [29]. The internal architecture of this family consists of two main blocks (see Fig. 9): the processor subsystem and a large FPGA. The processor subsystem contains a 32-bit ARM922T hard processor core, a memory subsystem, external memory interfaces, and standard peripherals, while the FPGA block consists of an APEX 20KE-like architecture with resources for SOPC integration. The bus architecture is based on the advanced microcontroller bus architecture (AMBA) high-performance bus (AHB). Nowadays, commercial CAD tools provided by FPGA vendors include limited user-friendly resources for HW/SW co-design, therefore, the development of SOPC-based solutions is a more complex task than the design of conventional FPGA-based approaches. Some of the problems that the designer has to tackle are:

- The definition of an efficient partition of the system into HW and SW blocks.
- The selection of an optimal word-length for the HW subsystem.
- The design of a high-performance HW architecture compatible with the transmission capabilities of the HW/SW communication interface.

Let us analyze these problems in the context of neuro-fuzzy modeling.

A. HW/SW Partition

One of factors that determines the suitability of HW/SW technologies to implement NFSs, is the well known distinctive characteristics of the main type of algorithms that a NFS involves, namely, the learning procedure and the feed-forward network. The learning procedure is a typical example of an algorithm better suited to SW implementations than to HW ones, due to three main factors, its inherent irregularity, its high computational demands, and its high precision requirements. On the other hand, the feed-forward network is a very regular and repetitive structure suitable for parallel HW implementation. Therefore, a suitable HW/SW partition consists in implementing the feed-forward network (7)-(9) as HW on the FPGA while the hybrid learning algorithms (10)-(13) and the input/output processing are implemented as SW on the microprocessor (ARM). The proposed HW/SW partition favours high-speed for real-time operation and exploits the resources of both the ARM and the FPGA. More precisely, the high numerical precision of the ARM processor (32 bits) ensures the proper behaviour of the learning algorithms, while FPGA allows the development of high performance parallel architectures.

B. Word-length Selection

The numerical precision of the ARM processor is high enough to minimize the effects of finite precision computing (or quantization errors) on the learning performance of the system. However, the selection of a suitable word-length to implement the feed-forward network is to be carefully analyzed. On the one hand, as is well known, larger word-lengths reduce the quantization errors in digital HW. However, on the other hand, large word-lengths penalize parameters such as speed, complexity, and cost of the circuits. Therefore, an optimal trade-off must be made between HW precision and the whole system performance. Special attention is to be paid to the consequences of the quantization errors on the approximation capabilities of the NFS.

Most of the works reported in the literature concerning the approximation capabilities of neural networks and fuzzy systems do not refer to the numerical limitations inherent in the finite precision computation of digital HW. In fact, the property of universal approximation no longer holds if the numerical limitations of finite word-length are taken into account [42]. In what follows we will consider as "full precision" simulations those simulations performed using Matlab's 64-bit floating-point arithmetic. Note that even full Matlab precision introduces numerical limitations. However, these limitations, as well as those involved in any present SW solution, can be considered irrelevant in the context of most neuro-fuzzy practical applications.

Finite precision errors are introduced in digital NFSs due to the quantization of both signals and parameters [43]. The first type of errors are the A/D (analogue to digital) errors where the samples of the analogue input signals are to be represented using a finite word-length. The second type of quantization errors are the membership function errors that result from the transfer and storage of the network parameters. Finally, the third type of errors are the arithmetic errors in the finite precision implementation of the algorithms. To analyze the consequences of the quantization errors on the approximation capability of the PWM ANFIS we will use the nonlinear functions (14) and (15). In the previous Section, the PWM ANFIS was trained to approximate these functions. Now, we are going to evaluate the previously trained feed-forward networks, but using finite precision computation. The experiments will be performed by means of the block scheme reported in Fig. 10. The quantization of the network parameters of the digitized PWM ANFIS will be performed by rounding their full precision values to finite word-length ones. To evaluate the network output for different word-lengths, we feed uniform distributed input samples to the network inputs (x_1^0, x_2^0). Then, we calculate the MSE errors between the full precision PWM ANFIS and the digitized PWM ANFIS by using different word-lengths. The simulation results are shown in Fig. 11. It can be seen that both experiment 1 and experiment 2 exhibit the same behaviour, the MSE is exponentially reduced as the word-length, *B*, increases. However, note that the errors provided by a word-length of 8 bits are comparable to the GMSE errors listed in Table I. Therefore, the approximation precision will not be improved even though greater word-lengths are used. As a consequence, the selection of oversized word-lengths (greater than 8 or 10 bits) to implement the feed-forward network will produce a useless precision excess and, on the other hand, word-lengths of less than 6 bits will give poor approximations. In view of these results and taking into account that the simplicity of digital HW design is greatly enhanced by selecting power-of-two word-lengths, an internal word-length of 8 bits is a good option. This selection has additional advantages, such as the compatibility with standard peripherals.

C. HW/SW Interaction

Another factor that affects the HW efficiency in a HW/SW approach is the communication overload between the microprocessor and the HW block. To avoid this kind of problem, let us analyze the transfer rates of input data and network parameters that are required to properly take advantage of a certain degree of parallelism in the HW implementation of the feed-forward PWM ANFIS. The analysis of the HW/SW interaction will be performed in terms of some useful design rates for neural network implementations defined in [44]. First, the data (or parameters) transfer rate of layer l, R_l, with $1 \le l \le 3$, is defined as follows

$$R_l = \frac{I_l}{T_l}(bits/s), \tag{16}$$

being I_l the number of I/O bits of layer l, and T_l the time required to compute layer l. Consider an n-input single-output PWM ANFIS and let B_I and B_P be the number of bits used to represent each system input and each parameter respectively. Also let B_{O1}, B_{O2}, and B_O be the number of bits dedicated to represent each node output in layer 1, layer 2, and layer 3 respectively (refer to Fig. 3). The first layer evaluates n membership functions (one per input dimension) and their corresponding fuzzy complement. Each node function in this layer is $O_i^{(1)} = f(x_i^0; a_i^{r_i^0}, b_i^{r_i^0})$, where x_i^0 is represented by means of B_I bits, and both $a_i^{r_i^0}$ and $b_i^{r_i^0}$ parameters are B_P–bit words; although the antecedents are trained by adjusting only the triangle offsets $(b_i^{r_i})$, after training is completed, the slopes of the triangles $(a_i^{r_i})$ are also computed to simplify further processing. Therefore, layer 1 involves $nB_I + 2nB_P$ input bits and $2nB_{O1}$ output bits. As a consequence, a total of $n(B_I + 2B_P + 2B_{O1})$ input/output (I/O) bits must be transferred in this layer. Using similar considerations, the number of I/O bits in layer 2, which has no parameter entries, is $2nB_{O1} + 2^n B_{O2}$, where the first term accounts for input bits and the second one accounts for the output bits of the 2^n nodes of this layer. Finally, layer 3 receives 2^n signals from the previous layer plus 2^n consequent parameters, and generates a single output, that is a total I/O bits of $2^n(B_{O2} + B_P) + B_O$. These results have been summarized in Table II.

The time required to process each layer in (16), T_l, can be calculated as follows

$$T_l = \frac{NN_l}{P_l} n_l T_{CLK}, \tag{17}$$

where NN_l is the number of neurons in layer l, T_{CLK} is the clock period, n_l is the number of clock cycles required to compute each neuron, and P_l is the number of processing units per layer. The number of neurons per layer is $NN_{1,2,3}=(2n,2^n,1)$, while the rest of the parameters depend on the concrete system architecture and on the target technology. With the above information, we evaluated the data transfer rate and parameter transfer rate (16) for the PWM ANFIS (refer to Table II). These results must be taken into account in the design of the HW partition. Since the availability of I/O bits is limited because of the limited parallelism of the HW/SW interface, the parallelism of the HW partition must be carefully dimensioned. An excess of parallelism produces a communication bottleneck that gives rise to inefficient developments.

Let us analyze the HW/SW interaction in the implementation of the SOPC-based PWM ANFIS, where the HW/SW interface consists of a finite-width bus (a 32-bit AMBA AHB bus). To be more specific, consider the suitability of this technology for implementing an architecture with intensive HW/SW communication demands as is the case of a fully parallel architecture. In this architecture each layer features as many processing elements as neurons in that layer ($P_1=2n, P_2=2^n, P_3=1$). In addition, assume that each processing element performs its operation in a single clock cycle ($n_1=n_2=n_3=1$); note that this requires processing elements that process binary words in parallel. The required transfer rates for this particular case are shown in Table III. Since the complete feed-forward network will be implemented as HW on the FPGA, the most critical layers are those that interface with the SW partition by means of the 32-bit bus. Layer 2, which is an internal layer, is not expected to present communication overload because a large amount of communication channels between FPGA cells is available. In view of the results shown in Table III, it is evident that the major communication problems are the transfer of $nB_I + 2nB_P$ input bits per clock cycle that requires the first layer to perform parallel computation efficiently, and the $2^n B_P$ input bits per clock cycle involved in the computation of layer 3. Let us suppose that the processor subsystem has to supply all these bits to the FPGA block by means of the 32-bit width bus. In this case, by selecting an 8-bit word-length for both data and parameters ($B_I=B_P=8$), the required transfer rate (data plus parameters) is of $24n$ bits/T_{CLK} in the first clock cycle (layer 1), and 2^{n+3} bits/T_{CLK} in the third clock cycle (layer 3). This data parallelism would be possible only in the case of a single-input system ($n=1$). A two-input system ($n=2$) requires 48 bits/T_{CLK} to feed the first layer; this exceeds the 32 bits of the bus. Therefore, a fully parallel system is not efficient whenever the system parameters and the input data have to be transferred by means of the HW/SW interface. In this sense, there are two possible alternatives to design efficient HW/SW solutions. The first one consists in storing the system parameters in the FPGA part of the device (HW partition) in order to have full access to them, while the second solution consists in designing HW architectures with a degree of parallelism tailored to the bit transfer rate that the HW/SW interface is able to provide. Both approaches will be considered in the next Section.

D. *Efficiency measure of HW/SW implementation*

Another aspect of the development of HW/SW solutions that is to be assessed is the efficiency of the HW/SW partition against a purely SW approach. Let us define the following efficiency index,

$$\eta = \frac{\tau_{SW}}{\tau_{HW/SW}} \tag{18}$$

where τ_{SW} is the time required to compute a certain process using a SW approach and $\tau_{HW/SW}$ accounts for the time required to evaluate the same process but using a concrete HW/SW implementation whose efficiency we are interested in evaluating. We will consider that the proposed HW/SW implementation is efficient to implement that process if $\tau_{HW/SW}<\tau_{SW}$, that is, if $\eta>1$. Equation (18) can be rewritten as follows,

$$\eta = \frac{T_{SW} n_{SW}}{T_{SW} n'_{SW} + T_{HW} n'_{HW}} = \frac{n_{SW}}{n'_{SW} + (f_{SW}/f_{HW}) n'_{HW}} \tag{19}$$

where T_{SW} (f_{SW}) and T_{HW} (f_{HW}) are the clock periods (frequencies) of the SW and the HW processors respectively, n_{SW} is the number of clock cycles required to compute the full process using SW, and n'_{SW} and n'_{HW} account respectively for the SW clock cycles and the HW clock cycles that are involved in a HW/SW implementation of the same process.

Table IV shows n_{SW} involved in the computation of the feed-forward PWM ANFIS using a one cycle per instruction processor like the ARM embedded in the Excalibur device. For simplicity, it has been assumed that every input space has been partitioned into the same number of membership functions, $NA_i=NA$, with $1\leq i\leq n$. Although the computation of the three layers of the network is independent of this parameter, a previous pre-processing step is required to localize the active cell and its parameters (offset, slope, and consequents), and to subtract the offset from the input to represent it in coordinates of the active interval (local coordinates). This step requires the comparison of each input with the NA-2 offsets for which it is necessary to load each offset in a register and to do n differences. Layer 1 then performs sequentially the following operations: the product of the local input value by the slope, which has also to be previously loaded into a register, and the logical complement of the product in order to obtain the complementary membership value. Layer 2 performs as many as $2^n(n\text{-}1)$ product operations, that is, n-1 products per each active rule. Finally, layer 3 can be efficiently processed by means of 2^n multiply-accumulate (MLA) operations and the same number of consequent loads from RAM to registers. As summarized Table IV, the number of clock cycles involved in a SW computation of the feed-forward network grows exponentially with the number of system inputs. Therefore, it is expected that for large multidimensional systems even HW architectures with a low degree of parallelism (compatible with the transfer rate of the HW/SW interface) are able to outperform a SW based solution. In view of these considerations and the results discussed in the previous Subsections, we propose two efficient HW/SW architectures to implement the PWM ANFIS. Firstly, a high speed parallel architecture that stores the network parameters in the FPGA part of the chip, after the SW training process is completed (off-line learning). Secondly, a pipelined architecture tailored to the transfer rate provided by the HW/SW interface and suitable for on-line parameter update.

V. HW/SW IMPLEMENTATIONS OF THE PWM ANFIS

A. *Design methodology*

In this work, a semi-automatic design methodology has been applied to support the development of the HW/SW-based PWM-ANFIS. First, the PWM-ANFIS Toolbox for Matlab, developed by the authors, has been used for full-precision and finite-precision computer simulations. The system has been divided into four main blocks with a well defined functionality: the feed-forward PWM ANFIS, the LSE algorithm, the GDM, and the input-output processing. A functional C model of each block has been compiled. In the next step, according to the HW/SW partition proposed in Section IV, both the HW and SW blocks have been gradually refined using computer-aided design tools. GNUPro tools for ARM have been used to develop the SW blocks, whereas Altera´s Quartus II design SW has been used to design the FPGA-target blocks, perform the HW/SW integration, perform simulations at the bus-transaction level, and configure the device. In the following, two efficient HW/SW variants, developed by the authors, will be presented.

B. *Fully Parallel Architecture*

In the previous Section we have demonstrated that HW architectures with a high degree of parallelism are efficient only if the network parameters are stored in the HW partition of the SOPC (FPGA block). Otherwise, the limited bandwidth of the HW/SW interface reduces the efficiency of parallel architectures. Our first proposal consists in a high-speed parallel architecture where the network parameters are stored in the FPGA part of the device. The proposed architecture operates in two stages, the training stage (off-line learning) and the feed-forward stage (on-line processing). First, the SW partition performs the parameter identification of the network using a set of training patterns as has been explained in Section II. After that, the adjusted parameters are transferred to the FPGA block and stored in a dedicated memory. If NA_i is the number of antecedents per input, $1\leq i\leq n$, the information transferred to the HW partition after training consists of $2\sum_i (NA_i - 1)$ antecedent parameters (two parameters per region), and $\prod_i NA_i$ consequent parameters (one consequent per rule). In the second stage, the HW partition performs the feed-forward processing of the PWM ANFIS each time a new input is presented. Fig. 12 depicts the block scheme of the fully parallel architecture for the case of an n-input system. The proposed architecture consists of four main blocks: the parameter memory, the antecedent multiplexer (AMUX), the consequent multiplexer (CMUX), and the neural network unit (NNU).

The parameter memory stores the network parameters. It consists of one 8-bit register per parameter and is equipped with a very flexible interconnection scheme that allows full parallel access to the memory contents. The size of the memory depends on the system inputs and also on the number of antecedents per input as has been explained above. The AMUX multiplexes the contents of the parameter memory and selects for transmission the parameters of the active region. The selection signals of the AMUX are the system inputs. The outputs of the module are the system inputs but expressed in coordinates of the active region $(x_i^{r_i^0})$, the region slope $(a_i^{r_i^0})$, and a selection signal that drives the CMUX. The CMUX selects from the parameter memory those consequents related to the active rules (c_j^0). The AMUX consists of *n* instances (one per input) of a VHDL component called COMP_SELECT (see Fig. 13). The architecture body of this component consists of a single process statement that is sensitive to all the input signals of the entity (i.e., the process defines combinational logic). A conditional statement (IF statement) compares the input x_i^0 with the region offsets $b_i^{r_i}$ until the condition $(x_i^0 \geq b_i^{r_i})$ becomes true. In this situation, the active region has been found and the entity outputs are: a pointer to the active region r_i^0, the slope of the active region $a_i^{r_i^0}$, and the difference between the input and the region offset, $x_i^{r_i^0} = x_i^0 - b_i^{r_i^0}$. The active region pointers are combined by means of the concatenate operator to construct the *Select* signal, as can be seen in Fig. 13. The CMUX selects the active consequents (c_j^0) from the parameter memory. It is a single-cycle VHDL component. The behaviour of the CMUX is given by a single process, activated by the system clock, that encloses a conditional statement (CASE statement). The *Select* signal, generated by the AMUX, drives the CASE statement in the selection of the active consequents. The NNU implements the three layers of the feed-forward network. It consists of a parallel architecture organized into three layers, as in Fig. 3. Layer 1 features one two-input multiplier per input. The multipliers used in this layer provide, in a single clock cycle, active-high output and active-low output to implement the pairs of complementary neurons. The second layer is composed of one *n*-input multiplier per rule, that is, 2^n multipliers. The product of *n* signals can be performed by means of 2-input multipliers organized into a typical tree-like structure as can be seen in Fig. 14. The number of clock cycles required to evaluate the product of *n* signals is $\log_2 n$. If *n* is not a power of two, the next power of two is to be used. For example, the evaluation of the tree of products involves 1 clock cycle if *n*=2, 2 clock cycles if *n*=4 (or *n*=3), 3 clock cycles if *n*=8 (or *n*=5, 6, 7), and so on. Finally, layer 3 of the NNU consists of 2^n two-input multipliers and a parallel *n*-input adder. This layer performs the sum of products in 2 clock cycles. Table V summarizes the number of clock cycles involved in the computation of this HW partition (n'_{HW}).

1) Efficiency Evaluation: Let us analyze the efficiency of the proposed architecture to implement the feed-forward network, against a purely SW solution in the sense given by efficiency index (18). Note that the fully parallel architecture implements the whole feed-forward network as HW on the FPGA, therefore $n'_{SW} = 0$. Therefore, the expression of the efficiency index given in (19), applied to the feed-forward network, can be rewritten as follows, $\eta = (n_{SW} f_{HW})/(n'_{HW} f_{SW})$, where n_{SW} and n'_{HW} can be found in Table IV and Table V, respectively. Finally, the efficiency of this implementation is,

$$\eta_{Parallel} = \frac{2^n(n+1)+2nNA}{4+\log_2 n}(f_{HW} / f_{SW}) \qquad (20)$$

Fig. 15 shows the efficiency index (20) as a function of the number of system inputs for different values of f=f_{HW}/f_{SW}; a frequency range 40 MHz ≤ f_{HW} ≤ 100 MHz, compatible with the target FPGA technology has been selected for evaluation. A mean value for the number of antecedents has been assumed to be *NA*=4. Taking into account that the maximum frequency specified for the embedded ARM processor is f_{SW}=200MHz, it can be concluded that the proposed parallel architecture is efficient against a SW implementation for frequencies f_{HW} greater than 40 MHz (f=0.2), except for the case of a single input that requires at least f_{HW}=66.6 MHz to be efficient, however this case is not significant. As was expected, the efficiency index grows exponentially as the number of inputs increases. Therefore, systems with a large number of inputs, which implies a reduction of the operation frequency or some additional clock cycle in the processing of the last layer, can also be efficiently implemented.

2) System Prototyping: As a case example of the proposed HW/SW approach, we developed a two-input PWM ANFIS. The object code, optimized for ARM922T, is stored in the internal single-port SRAM occupying approximately 10 Kbytes of the 32 Kbytes available in the device. The ARM processor executes the SW part (off-line training) of the PWM ANFIS operating up to 200 MHz; this performance is uncompromised by the FPGA operation. The HW partition, implemented in the FPGA, operates as a slave of the ARM processor and performs the feed-forward PWM ANFIS in only five clock cycles (see Fig. 12). The implementation of the HW partition has been carried out using VHDL descriptions and Altera´s macrofunctions. After synthesis, the HW part uses 2447 of the 4160 logic elements available in the FPGA part of the EPXA1F484C1 (58%) and allows a maximum clock frequency of 67 MHz. After truncation of the least significant bits of layer 3, the system output is given in a 32-bit two-complement format. The system has been successfully implemented and tested using the EPXA1 development board [45].

C. *Pipeline architecture*

The second approach is intended for on-line parameter learning. The same main HW/SW partition as for the parallel architecture will be used, that is, the learning algorithms will be developed in the SW partition and the feed-forward network will be implemented in the HW part of the device. The feed-forward network consists of a pipelined architecture designed to take full advantage of the transfer rate allowed by the HW/SW interface. Since the pipeline architecture performs on-line parameter adaptation, the parameter memory is to be located in the SW partition to avoid HW/SW communication delays.

The pipeline architecture has been designed taking into account the transfer rates analyzed in Section IV. It consists of a two-pipeline structure designed to increase the performance of the circuit. After analyzing carefully the efficiency of different solutions compatible with the available bit transfer rate, we decided to move the pre-processing step of the feed-forward network to the SW partition. In the pre-processing step the active parameters $(a_i^{r_i^0}, b_i^{r_i^0})$ are located, and the inputs (x_i^0), $1 \le i \le n$, are represented in coordinates of the active region (r_i^0), that is, $x_i^{r_i^0} = x_i^0 - b_i^{r_i^0}$. After the pre-processing step, the triangle offsets $(b_i^{r_i^0})$ are not required to complete the computation of the feed-forward network. The advantage of implementing the pre-processing step in the SW partition is that only one antecedent parameter per input, the triangle slope $(a_i^{r_i^0})$, is to be transmitted through the bus, instead of the pair $(a_i^{r_i^0}, b_i^{r_i^0})$ that would be required to implement the pre-processing step in the HW partition. The authors also investigated the efficiency of different solutions where the pre-processing step was implemented in the HW partition, but in all the studied cases, the delay in the transmission of data through the bus gave rise to non efficient HW/SW solutions. Let us now present a detailed description of the pipeline architecture.

1) HW/SW Co-operation and Data Transactions: The HW/SW co-operation inside the device is based on the AMBA AHB bus architecture and also on its associated embedded bridges [46]. The ARM is the bus master in the processor subsystem (PS) and has fast access to the FPGA slaves (feed-forward network) via the PS-to-FPGA bridge. (see Fig. 16). The main data flow across the HW/SW interface in the pipeline architecture is the transmission of $2n + 2^n$ bytes per inference from the PS to the FPGA (n inputs, n slopes and 2^n consequents); the opposite data flow is a single 32-bit word per inference (system output). The transaction throughput and the co-operation performance depends on the interface configuration and also on the relative clock speed of the master domain and the slave domain. Below in this Section we will give concrete results of the transaction throughput for a particular case example.

Since the bus that communicates the PS and the FPGA block is 32-bit width, the data transmission can be straightforwardly organized into four bytes per transfer, as has been depicted in Fig. 17. The data transfer from SW to HW starts by sending each local input and its associated active slope $(x_i^{r_i^0}, a_i^{r_i^0})$. Two data pairs occupy 32 bits, therefore, an n-input system requires $n/2$ transfers to perform the transmission of all the data involved in the processing of the first layer of the feed-forward network (if n is odd, the next even number is to be used to compute these transfers). The second layer has no additional inputs, while the computation of layer 3 involves 2^n consequent parameters that have been arranged in packets of four consequents each, so that we need 2^{n-2} transfers to complete the transmission of the 2^n consequents.

2) Pipeline Structure: Fig. 18 depicts a block scheme of the pipeline architecture for the case of an n-input PWM ANFIS. The proposed architecture performs the same computations as the neural network unit (NNU) of the parallel architecture (see Fig. 12), previously presented, but limiting the degree of parallelism exactly to the availability of data. As can be seen, the architecture has been structured into two pipelines. The synchronization of the two pipelines is carried out by a simple control unit. This unit generates the signals for the pipeline control and also generates the signals required to synchronize with the SW partition.

The data path of the first pipeline consists of three single-cycle stages: the membership function stage, the partial rule activation stage, and the register stage. The first stage has to be able to process the two inputs provided in each transfer concurrently. This is accomplished by two parallel two-input multipliers. The multipliers provide both active-high and active-low outputs to implement the fuzzy complements in (7). The next stage of the pipeline computes four partial rule activations concurrently by means of four parallel two-input multipliers. We say that the activation is partial because it considers only the interaction (product) between pairs of inputs in (8). In the last stage of this pipeline, the partial rule activations are stored (four each clock cycle) into a battery of $2n$ registers. The sequence of register loads is controlled by means of $n/2$ enabling signals, as can be seen in Fig. 18.

Since each pair of inputs requires two cycles to complete the HW/SW data transfer, the above three-stage pipeline requires one wait cycle before the next pair of inputs enters the pipeline. Therefore, the total number of clock cycles required to compute the first pipeline, including data transfers, is equal to the pipeline length (3 cycles), plus the number of words to be processed ($n/2$) multiplied by the input delay (2 clock cycles), that is,

$$n_{pipeline-1} = 3 + n \tag{21}$$

The second pipeline performs the rest of the feed-forward network once the first pipeline finishes. The sequence of operations involved in the second pipeline has been organized into ($3+\log_2 n$) stages. In the first stage, the rule activation multiplexer (RAMUX) selects four different combinations of the $2n$ partial rule activations computed in the first pipeline.

The RAMUX is a single-cycle VHDL component. It consists of a single process activated by the system clock. A conditional statement (CASE statement) multiplexes the partial rule activations; the selection signal is generated by a counter in the control unit. The outputs of the RAMUX are the inputs to the next layer of the circuit that consists of four parallel (n/2)-input multipliers. To avoid excessive signal delays, each multiplier has been implemented using two-input multipliers interconnected in the form of a binary tree structure (see Fig. 14), as in the previous fully parallel architecture. The number of clock cycles or stages required to evaluate the binary tree is equal to $\log_2(n/2)$ or equivalently, $\log_2 n - 1$ (if n is not a power of two, the next power of two is to be used). The next stage of the pipeline receives four consequents and four rule activations, and performs the four products in parallel. After that, a four-input parallel adder performs the partial sum of four output terms per clock cycle. Finally, in the last stage of the pipeline, the partial sum of products is accumulated. A new group of four consequents enters the pipeline every two clock cycles, until the transmission of the 2^n consequents is completed (2^{n-2} 32-bit words). Therefore, the total clock cycles required for the circuit to evaluate the second pipeline is: the length of the pipeline ($3+\log_2 n$) plus the product of the number of words (2^{n-2}-1) per their input delay (2 cycles), where the first word transfer has no wait cycles because it is performed while the first pipeline is still active,

$$n_{pipeline-2} = 1 + \log_2 n + 2^{n-1}\,. \tag{22}$$

In summary, the HW part of the feed-forward network involves (21) plus (22) HW clock cycles, that is, $n'_{HW} = 4 + n + \log_2 n + 2^{n-1}$ clock cycles. This information is presented in Table VI and will be used to compute the efficiency of the pipeline architecture.

3) Efficiency Evaluation and System Prototyping: The efficiency index (19) applied to the pipeline architecture results,

$$\eta_{pipeline} = \frac{2^n(n+1) + 2nNA}{n(2NA-3) + [4 + n + \log_2 n + 2^{n-1}]/(f_{HW}/f_{SW})}, \tag{23}$$

where the numerator, n_{SW}, has been obtained from Table IV, and the terms of the denominator, n'_{SW} and n'_{HW}, can be found in Table VI. Fig. 19 shows the efficiency index (23) as a function of the number of inputs, for different values of $f=f_{HW}/f_{SW}$. As for the parallel architecture, a frequency range 40 MHz $\le f_{HW} \le$ 100 MHz has been used and a mean value for the number of antecedents NA=4 has been assumed. Taking into account that the maximum frequency specified for the embedded ARM processor is f_{SW}=200MHz, it can be concluded that the pipeline architecture is efficient against a SW implementation for PWM ANFIS with three or more inputs. In the case of a 3-input system, frequencies f_{HW} greater than 60 MHz (f=0.3) are required to develop efficient implementations. Note that the efficiency index grows as the number of inputs increases, but slowly than in Fig. 15.

As a prototyping example, we developed the pipeline architecture of a four-input PWM ANFIS. The design has been developed using the EPXA4F672C1 device of Altera´s Excalibur family. The EPXA4 devices are the second in size (400.000 typical gates) after the EPXA1 devices (100.000 typical gates) used to implement the two-input fully parallel architecture. After synthesis, our design uses 4.234 of the 16.640 logic elements available in the FPGA part of the SOPC (25%). Note that the implementation of larger systems, with more than four inputs, does not imply an important increase of resources; only the register stage and the RAMUX have to be modified. The ARM processor executes the SW part of the system operating up to 200 MHz, while the HW partition, implemented in the FPGA, performs the feed-forward network with a maximum clock frequency of 50.3 MHz. This clock domain distribution leads to a transaction throughput between the ARM and the FPGA of approximately 100 Mbytes per second (see Fig. 16). The transmission of 24 bytes from the processor subsystem (PS) to the FPGA (n inputs, n slopes, 2^n consequents), and 4 bytes back from the FPGA to the PS (32-bit word that represents the system output) requires only 0.28 μs, while a complete inference takes 0.358 μs (18 cycles with a 50.3 MHz clock, see Table VI). The above operation frequencies give an efficiency index of 1.4 in (23). In view of these results, it can be concluded that the HW/SW interface is fast enough to guarantee the suitability of the proposed HW/SW architecture for the four-input PWM ANFIS.

VI. Conclusion

In this work we have reported the development of two on-chip HW/SW architectures for a particular case of NFS suitable for real-time embedded applications. The NFS consists of an ANFIS-like model modified for efficient HW/SW implementation. As a consequence of some constraints imposed on the model, the complexity of the feed-forward network and the learning algorithms are greatly reduced. In particular, the feed-forward network becomes a piecewise multilinear (PWM) function with a simple cellular structure. It has been verified that the PWM ANFIS, in spite of the restrictions, exhibits approximation capabilities and learning abilities comparable to those of generic ANFIS.

To develop efficient architectures for the PWM ANFIS, it is necessary to consider first some important problems that arise in the development of HW/SW approaches: a) the selection of an efficient partition of the system into HW and SW blocks, b) the selection of an optimal word-length for the HW subsystem, and c) the design of high-performance HW architectures compatible with the transmission capabilities of the HW/SW communication interface. As a consequence of the previous analysis, two different on-chip HW/SW architectures have been developed. The first architecture consists of a high-speed parallel architecture with fixed parameters (off-line learning), while the second one is a pipelined architecture, tailored to the available HW/SW bit transfer rate, suitable for on-line parameter adaptation. Both solutions have been implemented on a SOPC of Altera's Excalibur family.

The main feature of the proposed solutions is a trade-off between versatility and performance. In this sense, the embedded processor provides flexibility and high precision to implement the hybrid learning algorithms, while the FPGA block provides high-speed to process the feed-forward neural network. The proposed approach is suitable for developing efficient implementations for already known application areas of embedded NFSs such as consumer electronics, robotics, or automotive control, among others. In addition, potential applications can be found in the context of pervasive computing applied to ambient intelligence [47]. Ambient intelligence needs small embedded systems able to deal on-line with a large number of inputs, and also able to adapt themselves to changing conditions and user preferences [48]. These requirements can be met by means of the SOPC-based PWM ANFIS presented in this work.

In future works we intend to enhance the capabilities of the neuro-fuzzy SOPC to cope with the problem of abrupt context changes. In these situations, the adaptation of the NFS to the new context would require structural changes in the feed-forward network, in addition to the parameter adaptation. This kind of structural changes involve the reconfiguration –total or partial– of the HW partition. In this sense, we will apply dynamic reconfiguration techniques to reconfigure the FPGA part of the device.

APPENDIX

The learning rule for $b_i^{r_i}$ is

$b_i^{r_i(t+1)} = b_i^{r_i(t)} - \eta_{b_i}\left(\frac{\partial E}{\partial b_i^{r_i}}\right)$, with $E = \frac{1}{2K}\sum_{k=1}^{K} E_k$, where η_{b_i} is the learning rate and $E_k = (y_k - y_k^{'})^2$. Using the chain rule, for each sample, *k*:

$$\frac{\partial E_k}{\partial b_i^{r_i}} = \frac{\partial E_k}{\partial y}\frac{\partial y}{\partial b_i^{r_i}} ,$$

$$\frac{\partial E_k}{\partial y} = 2(y - y^{'}) .$$

Referring to (9), $y = \sum_{j=0}^{2^n-1} w_j c_j^0$, therefore,

$\frac{\partial y}{\partial b_i^{r_i}} = \sum_{j=0}^{2^n-1} c_j^0 \frac{\partial w_j}{\partial b_i^{r_i^0}}$, applying once again the chain rule,

$\frac{\partial w_j}{\partial b_i^{r_i^0}} = \frac{\partial w_j}{\partial \phi_{j_i}}\frac{\partial \phi_{j_i}}{\partial b_i^{r_i^0}}$, since $w_j = \prod_{i=1}^{n} \phi_{j_i}$ (see (8)),

$$\frac{\partial w_j}{\partial b_i^{r_i^0}} = \frac{w_j}{\phi_{j_i}}\frac{\partial \phi_{j_i}}{\partial b_i^{r_i^0}} .$$

Using the definition of the triangular membership functions as a function of the adjustable parameters (offsets) given in Section II,

$\phi_{j_i} = \begin{cases} \mu_i^{r_i^0}(x_i^0) = (x_i^0 - b_i^{r_i^0})/(b_i^{r_i^0+1} - b_i^{r_i^0}), \text{ if } j_i = 1 \\ 1 - \mu_i^{r_i^0}(x_i^0) = -(x_i^0 - b_i^{r_i^0+1})/(b_i^{r_i^0+1} - b_i^{r_i^0}), \text{ if } j_i = 0 \end{cases}$, with $b_i^{r_i^0} \le x_i^0 \le b_i^{r_i^0+1}$. Note that the slope of the active triangles is $a_i^{r_i^0} = 1/(b_i^{r_i^0+1} - b_i^{r_i^0})$ (see Fig. 4).

$$\frac{\partial \phi_{j_i}}{\partial b_i^{r_i^0}} = \begin{cases} [-(b_i^{r_i^0+1} - b_i^{r_i^0}) + (x_i^0 - b_i^{r_i^0})]/(b_i^{r_i^0+1} - b_i^{r_i^0})^2, \text{ if } j_i = 1 \\ -(x_i^0 - b_i^{r_i^0+1})/(b_i^{r_i^0+1} - b_i^{r_i^0})^2, \text{ if } j_i = 0 \end{cases} ,$$

$\frac{\partial y}{\partial b_i^{r_i}} = \sum_{j=0}^{2^n-1} c_j^0 w_j f_{j_i}^{r_i^0}$, where

$$f_{j_i}^{r_i^0} = \frac{1}{\phi_{j_i}}\frac{\partial \phi_{j_i}}{\partial b_i^{r_i^0}} = \begin{cases} -1/(x_i^0 - b_i^{r_i^0}) + 1/(b_i^{r_i^0+1} - b_i^{r_i^0}), \text{ if } j_i = 1 \\ 1/(b_i^{r_i^0+1} - b_i^{r_i^0}), \text{ if } j_i = 0 \end{cases} .$$

Finally, summing all the training samples,

$$\frac{\partial E}{\partial b_i^{r_i}} = \frac{1}{K}\sum_{k=1}^{K}(y_k - y_k^{'})\sum_{j=0}^{2^n-1} c_j^0 w_j f_{j_i}^{r_i^0} .$$

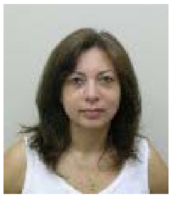
ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their useful comments and suggestions.

REFERENCES

[1] J. J. Buckley, "Sugeno type controllers are universal controllers," Fuzzy Sets Syst., vol. 53, no. 3, pp. 299–303, Feb. 1993.
[2] S. G. Cao, N. W. Rees, and G. Feng, "Mamdani-type fuzzy controllers are universal fuzzy controllers," Fuzzy Sets Syst., vol. 123, no. 3, pp. 359–367, Nov. 2001.
[3] B. Kosko, "Fuzzy systems as universal approximators," IEEE Trans. Computers, vol. 43, no. 11, pp. 1329–1333, Nov. 1994.
[4] R. Rovatti, "Fuzzy piecewise multilinear and piecewise linear systems as universal approximators in Sobolev norms," IEEE Trans. Fuzzy Systems, vol. 6, no. 2, pp. 235–249, May 1998.
[5] L.-X. Wang, and C. Wei, "Approximation accuracy of some neuro-fuzzy approaches," IEEE Trans. Fuzzy Systems, vol. 8, no. 4, pp. 470–478, Aug. 2000.
[6] X.-J. Zeng, and M. G. Singh, "Approximation theory of fuzzy systems-SISO case," IEEE Trans. Fuzzy Systems, vol. 2, no. 2, pp. 162–176, May 1994.
[7] X.-J. Zeng, and M. G. Singh, "Approximation theory of fuzzy systems-MIMO case," IEEE Trans. Fuzzy Systems, vol. 3, no. 2, pp. 219–235, May 1995.
[8] X.-J. Zeng, and M. G. Singh, "Approximation accuracy analysis of fuzzy systems as function approximators," IEEE Trans. Fuzzy Systems, vol. 4, no. 1, pp. 44–63, Feb. 1996.
[9] J.-S. R. Jang, C.-T Sun and E. Mizutani, Neuro-Fuzzy and Soft Computing. Upper Saddle River, NJ: Prentice Hall, 1997, part VII.
[10] A. Fanaei, and M. Farrokhi, "Robust adaptive neuro-fuzzy controller for hybrid position/force control of robot manipulators in contact with unknown environment," Journal of Intelligent & Fuzzy Systems, vol. 17, no. 2, pp. 125-144, 2006.
[11] S.-J. Ho, L.-S. Shu, and S.-Y. Ho, "Optimizing Fuzzy Neural Networks for Tuning PID Controllers Using an Orthogonal Simulated Annealing Algorithm OSA," IEEE Trans. Fuzzy Systems, vol. 14, no. 3, pp. 421-434, June 2006.
[12] H. B. Kazemian, and L. Meng, "An Adaptive Control for Video Transmission Over Bluetooth," IEEE Trans. Fuzzy Systems, vol. 14, no. 2, pp. 263-274, Apr. 2006.
[13] F.-J. Lin, and P.-H. Shen, "Adaptive Fuzzy-Neural-Network Control for a DSP-Based Permanent Magnet Linear Synchronous Motor Servo

Drive," IEEE Trans. Fuzzy Systems, vol. 14, no. 4, pp. 481-495, Aug. 2006.
[14] A. Rubaai, A. R. Ofoli, L. Burge, and M. Garuba, "Hardware Implementation of an Adaptive Network-Based Fuzzy Controller for DC-DC Converters," IEEE Trans. Industry Applications, vol. 41, no. 6, pp. 1557-1565, Nov/Dec. 2005.
[15] J. B. Theocharis, "A high-order recurrent neuro-fuzzy system with internal dynamics: Application to the adaptive noise cancellation," Fuzzy Sets Syst., vol. 157, no. 4, pp. 471-500, Feb. 2006.
[16] R.-J. Wai, and L.-J. Chang, "Stabilizing and Tracking Control of Nonlinear Dual-Axis Inverted-Pendulum System Using Fuzzy Neural Network," IEEE Trans. Fuzzy Systems, vol. 14, no. 1, pp. 145-168, Feb. 2006.
[17] M. E. Yüksel, and E. Besdok, "A Simple Neuro-Fuzzy Impulse Detector for Efficient Blur Reduction of Impulse Noise Removal Operators for Digital Images," IEEE Trans. Fuzzy Systems, vol. 12, no. 6, pp. 854-865, Dec. 2004.
[18] J. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," IEEE Trans. Systems, Man, and Cybernetics, vol. 23, no. 3, pp. 665-685, May 1993.
[19] J.-S. R. Jang, and C.-T. Sun, "Neuro-fuzzy modeling and control," Proceedings of the IEEE, vol. 83, no. 3, pp. 378-406, Mar.1995.
[20] M. Sugeno, and G. T. Kang, "Structure identification of fuzzy model," Fuzzy Sets Syst., vol. 28, no. 1, pp. 15–33, Oct. 1988.
[21] T. Takagi, and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," IEEE Trans. Systems, Man, and Cybernetics, vol. 15, no. 1, pp. 116-132, 1985.
[22] D. Anguita, I. Baturone, and J. Miller (Guest Editors), Applied Soft Computing. Special issue on hardware implementations of soft computing techniques, vol. 4, no. 3, Aug. 2004.
[23] B. Linares-Barranco, A. G. Andreou, G. Indiveri, and T. Shibata (Guest Editors), IEEE Trans. Neural Networks. Special issue on neural networks hardware implementation, vol. 14, no. 5, Sep. 2003.
[24] F. M. Salam, and T. Yamakawa (Guest Editors), Computers & Electrical Engineering. Special issue on micro-electronic hardware implementation of soft computing: neural and fuzzy networks with learning, vol. 25, no. 5, Sep. 1999.
[25] M. J. Patyra (Guest Editor), IEEE Trans. Fuzzy Systems. Special issue on fuzzy logic hardware implementations, vol. 4, no. 4, Nov. 1996.
[26] G. De Micheli (Guest Editor), Proceedings of the IEEE. Special issue on hardware/software codesign, vol. 85, no. 3, March 1997.
[27] W. Wolf (Guest Editor), "A decade of hardware/software codesign," Computer, vol. 36, no. 4, pp. 38-43, Apr. 2003.
[28] L. M. Reyneri, "Implementation issues of neuro-fuzzy hardware: going toward HW/SW codesign," IEEE Trans. Neural Networks, vol. 14, no. 1, pp. 176-194, Jan. 2003.
[29] Datasheet of Excalibur Device, Altera Corporation [Online]. Available: http://www.altera.com
[30] I. Baturone, S. Sánchez-Solano, A. Barriga, and J. L. Huertas, "Design issues for the VLSI implementation of universal approximator fuzzy systems," in Proc. World MultiConference on Circuits, Systems, Communications and Computers, Athens, 1999, pp. 6471–6476.
[31] I. Baturone, A. Barriga, S. Sánchez-Solano, C. J. Jiménez-Fernández, D. R. López. Microelectronic Design of Fuzzy Logic-based Systems. Boca Raton, Florida: CRC Press, 2000, ch. 9, 12.
[32] J. Echanobe, I. del Campo, and J. M. Tarela, "Issues concerning the analysis and implementation of a class of fuzzy controllers," Fuzzy Sets Syst., vol. 155, no. 2, pp. 252-271, Oct. 2005.
[33] R. Rovatti, C. Fantuzzi, and S. Simani, "High-speed DSP-based implementation of piecewise-affine and piecewise-quadratic fuzzy systems,"Signal Processing, vol. 80, no. 6, pp. 951-963, Jun. 2000.
[34] I. Baturone, A. Barriga, S. Sánchez-Solano, and J.L. Huertas, "Mixed-signal design of a fully parallel fuzzy processor," Electronics Letters, vol. 34, no. 5, pp. 437-438, March 1998.
[35] J. Matas, L. García de Vicuña, and M. Castilla, "A Synthesis of Fuzzy Control Surfaces in CMOS Technology," in Proc. IEEE Int. Conference on Fuzzy Systems, Barcelona, 1997, pp. 641–646.
[36] F. Vidal-Verdú, R. Navas-González, and A. Rodríguez-Vázquez, "Multiplexing architecture for mixed-signal CMOS fuzzy controllers," Electronics Letters, vol. 34, no. 14, pp. 1437-1439, Jul. 1998.
[37] J.-S. R. Jang, C.-T. Sun and E. Mizutani, Neuro-Fuzzy and Soft Computing., Upper Saddle River, NJ: Prentice Hall, 1997, part III, ch. 8.
[38] S.-J. Lee, and C.-S. Ouyang, "A neuro-fuzzy system modeling with self-constructing rule generation and hybrid SVD-based learning," IEEE Trans. Fuzzy Systems, vol. 11, no. 3, pp. 341-353, June. 2003.
[39] Y. H. Lin, G. A. Cunningham, and S. V. Coggeshall, "Using fuzzy partitions to create fuzzy systems from input-output data and set the initial weights in a fuzzy neural network," IEEE Trans. Fuzzy Systems, vol. 5, no. 4, pp. 614-621, Nov. 1997.
[40] C.-C. Wong, and C.-C. Chen, "A hybrid clustering and gradient descent approach for fuzzy modeling," IEEE Trans. Syst., Man, Cybern., B, vol. 29, no. 6, pp. 686-693, Dec. 1999.
[41] K. Basterretxea, J. M. Tarela, I. del Campo, and G. Bosque, "An experimental study on nonlinear function computation for neural/fuzzy hardware design," IEEE Trans. Neural Networks, vol. 18, no. 1, pp. 266-283, Jan. 2007.
[42] J. Wray, and G. G. Green, "Neural networks, approximation theory, and finite precision computation," Neural Networks, vol. 8, no.1, pp. 31-37, 1995.
[43] I. del Campo, and J.M. Tarela, "Consequences of the Discretization on the performance of a fuzzy logic controller," IEEE Trans. Fuzzy Systems, vol. 7, no.1, pp. 85-92, Feb. 1999.
[44] L. M. Reyneri, and F. Renga, "Speeding-up the design of HW/SW implementations of neuro-fuzzy systems using the CodeSimulink environment," Applied Soft Computing, vol. 4, no.3, pp. 227-240, Aug. 2004.
[45] EPXA1 Development Board. Hardware Reference Manual, Altera Corporation [Online]. Available: http://www.altera.com
[46] Excalibur Solutions-Using the Embedded Stripe Bridges. Application Note 142, Altera Corporation [Online]. Available: http://www.altera.com
[47] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman, "Scenarios for ambient intelligence in 2010," Tech. Rep., IST Advisory Group (ISTAG), Institutional Prospective Technology Studies (IPTS), Seville, Feb. 2001.
[48] F. Doctor, H. Hagras, and V. Callaghan, "A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments," IEEE Trans. Syst., Man, Cybern., A, vol. 35, no. 1, pp. 55-65, Jan. 2005.



Inés del Campo (M'94–A'96) was born in Buenos Aires, Argentina, in 1961. She received the Ph.D. degree in physics from the University of the Basque Country (UPV/EHU), Spain, in 1993.
Currently she is a Senior Lecturer in the Electricity and Electronics Department of the Faculty of Sciences and Technology of the UPV/EHU. Her research interests mainly concern hardware/software codesign, reconfigurable hardware, ANNs, fuzzy systems, and genetic algorithms.

Javier Echanobe (M'06) received the Licenciado degree in physics from the University of the Basque Country (UPV/EHU), Spain, and the Ph.D. degree from the University of Navarra, Pamplona, Spain, in 1990 and 1998, respectively.
He was a predoctoral researcher (granted by the Basque Government) from 1992 to 1996. Since 1999 he has been an Associate Professor in the Department of Electricity and Electronics of the UPV/EHU. His research interests focus on embedded systems, reconfigurable FPGAs, and computational intelligence.

José Manuel Tarela received the Licenciado degree in physics and the Ph.D. degree in physics from the Valladolid University, Spain, in 1970 and 1974, respectively.
Since 1976, he has been a Professor in the Faculty of Sciences and Technology, University of the Basque Country (UPV/EHU), Spain. He was the Director of the Electricity and Electronics Department and

Vicerrector of Investigation of the UPV/EHU.
Dr. Tarela is Member of the International Neural Network Society (INNS), the Real Sociedad Española de Física (RSEF), and the European Society fo Fuzzy Logic and Technology (EUSFLAT)

Guillermo Bosque received the Licenciado degree in physics with specialization in electronics and automatics in 1978 from the University of the Basque Country (UPV/EHU) , Bilbao, Spain.
From 1978 to 2002, he worked in several electronics industries, mainly in R&D management. From 2002 to 2003, he worked in a research group of the Electricity and Electronics Department at the UPV/EHU. Since 2003, he has been a Lecturer in the Department of Electronics and Telecommunication at the UPV/EHU.

FIGURE CAPTIONS

Fig. 1. Architecture of a generic *n*-input adaptive neuro-fuzzy inference system (ANFIS). The system is equivalent to the zero-order Sugeno inference model.

Fig. 2. Example of a two-input system verifying the constraints imposed on the general ANFIS model: the antecedents are triangles with an overlapping degree of two and normalized in each input dimension. The input vector (x_1^0, x_2^0) activates the highlighted semi-triangles that delimit the active cell.

Fig. 3. Architecture of the PWM-ANFIS. Grey neurons in layer 1 are *complementary* neurons with node function $\overline{O_i}^{(1)} = 1 - O_i^{(1)}$.

Fig. 4. Node function of the neurons in layer 1 of the PWM ANFIS.

Fig. 5. Experiment 1. (a) Graphical representation of the desired function. (b) Approximation of the desired function with a 16-rule PWM ANFIS. (c) Approximation of the desired function with a 36-rule PWM ANFIS.

Fig. 6. Experiment 1. Mean squared error (MSE) curves obtained with ANFIS (Gaussian membership functions) and with the PWM ANFIS (triangular membership functions).

Fig. 7. Experiment 2. (a) Graphical representation of the desired function. (b) Approximation of the desired function with a 9-rule PWM ANFIS.

Fig. 8. Experiment 2. Mean squared error (MSE) curves obtained with ANFIS (Gaussian membership functions) and with the PWM ANFIS (triangular membership functions).

Fig. 9. Internal architecture of the SOPC of Altera´s Excalibur family used to implement the PWM ANFIS, and partition of the system into hardware and software blocks.

Fig. 10. Block scheme used to evaluate the approximation capability of the PWM ANFIS when finite-precision computation is used.

Fig. 11. Mean squared error (MSE) between the full precision PWM ANFIS trained for experiments 1 and 2 and their digitized PWM ANFIS models for different word-lengths.

Fig. 12. Block scheme of the fully parallel architecture of an *n*-input PWM ANFIS implemented in the FPGA part of the SOPC.

Fig.13. Antecedent multiplexer (AMUX) module. The AMUX selects for transmission the parameters of the active regions and generates a *Select* signal that drives the Consequent MUX. The *Select* signal consists of the concatenation of the active region pointers; $Select = r_n^0 \,\&\cdots r_2^0 \,\&\, r_1^0$, where & is the concatenate operator.

Fig.14. Scheme of an 8-input multiplier implemented by means of two-input single-cycle multipliers. The multiplier is structured into a binary tree of 3 layers. In the general case, an *n*-input multiplier consists of $\log_2 n$ layers.

Fig. 15. Efficiency index (η) of the fully parallel architecture as a function of the number of system inputs for different values of $f=f_{HW}/f_{SW}$.

Fig. 16. Interface between the processor subsystem (PS) and the FPGA. The master port (M) initiates transactions and the slave port (S) responds to transactions. Clock frequencies in parenthesis correspond to the four-input PWM ANFIS (pipeline architecture).

Fig. 17. Organization of the data and parameter transferences in the pipeline architecture by using the 32-bit HW/SW interface.

Fig. 18. Block scheme of the pipeline architecture of an *n*-input PWM ANFIS implemented in the FPGA part of the SOPC.

Fig. 19. Efficiency index (η) of the pipeline architecture as a function of the number of system inputs for different values of $f=f_{HW}/f_{SW}$.

TABLE CAPTIONS

TABLE I
COMPARISON ON THE GENERALIZED MEAN SQUARED ERROR (GMSE) OF EXPERIMENT 1 AND EXPERIMENT 2

TABLE II
EVALUATION OF THE DATA TRANSFER RATE AND THE PARAMETER TRANSFER RATE FOR EACH LAYER OF THE PWM ANFIS

TABLE III
REQUIRED DATA TRANSFER RATE AND PARAMETER TRANSFER RATE FOR EACH LAYER OF A FULLY PARALLEL IMPLEMENTATION OF THE PWM ANFIS

TABLE IV
COMPUTATION OF THE FEED-FORWARD PWM ANFIS BY USING A ONE CYCLE PER INSTRUCTION PROCESSOR

TABLE V
COMPUTATION OF THE FEED-FORWARD PWM ANFIS BY MEANS OF THE FULLY PARALLEL ARCHITECTURE

TABLE VI
COMPUTATION OF THE FEED-FORWARD PWM ANFIS BY MEANS OF THE PIPELINE ARCHITECTURE

FIGURES

Figure 1

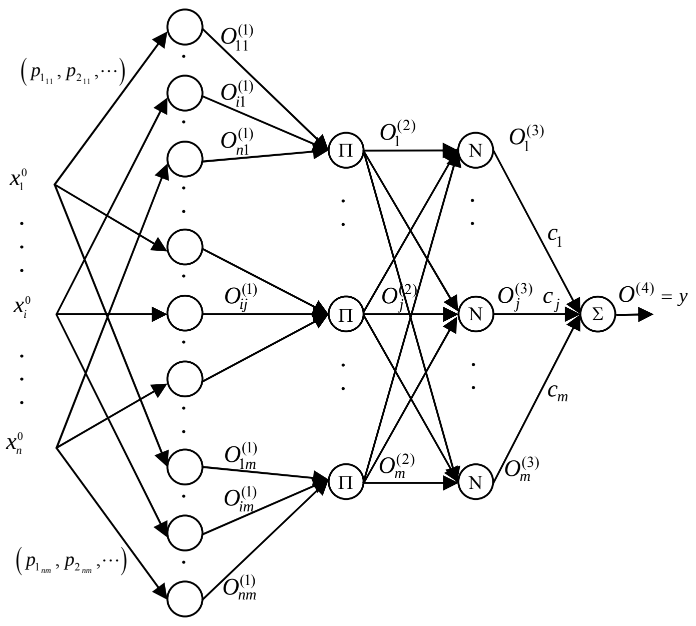


Figure 2

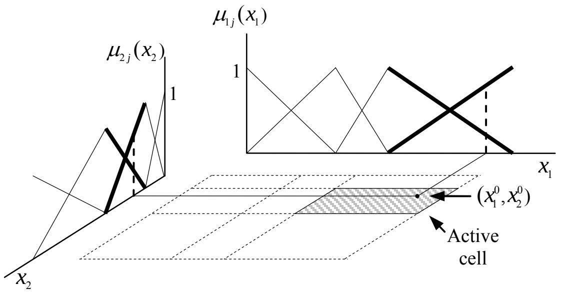


Figure 3

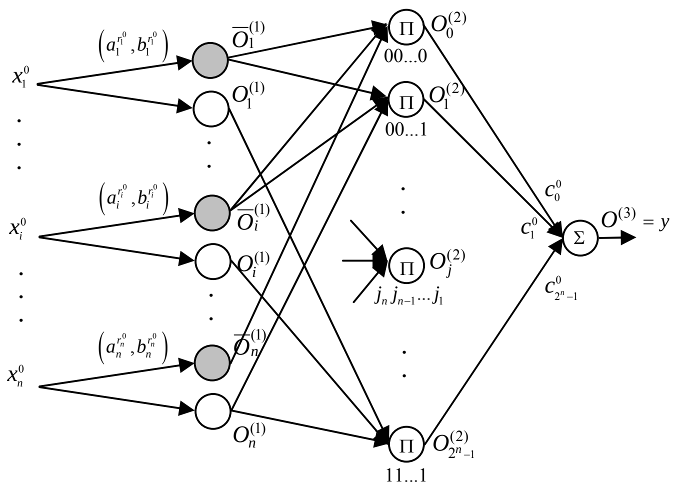
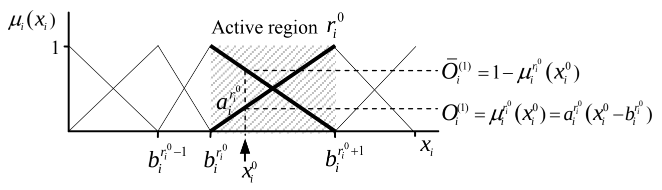


Figure 4



$\mu_i(x_i)$
1
Active region r_i^0
$\overline{O}_i^{(1)} = 1 - \mu_i^{r_i^0}(x_i^0)$
$O_i^{(1)} = \mu_i^{r_i^0}(x_i^0) = a_i^{r_i^0}(x_i^0 - b_i^{r_i^0})$
$a_i^{r_i^0}$
$b_i^{r_i^0-1}$
$b_i^{r_i^0}$
x_i^0
$b_i^{r_i^0+1}$
x_i

Figure 5

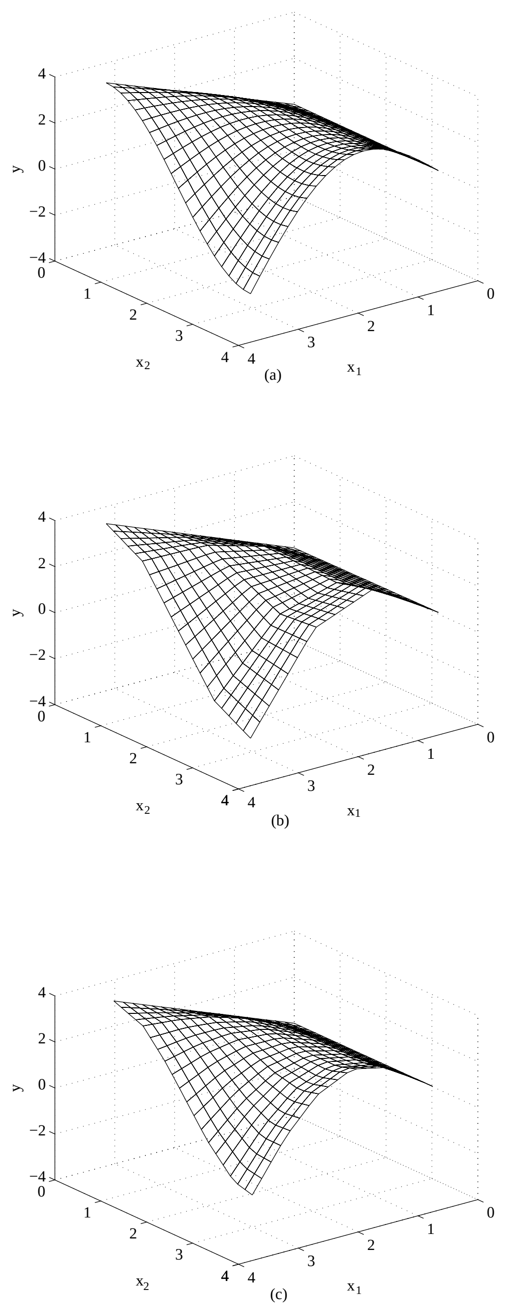


Figure 6

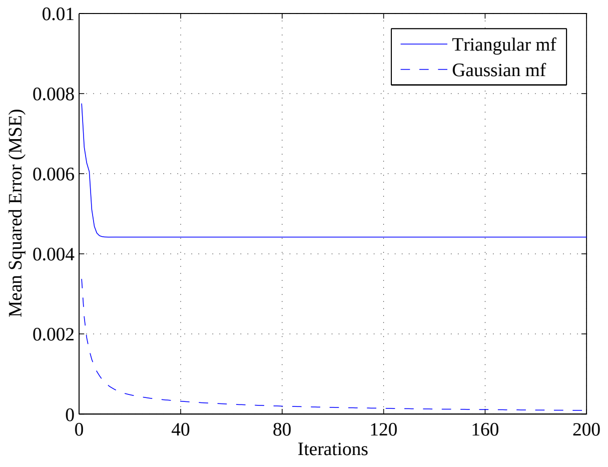
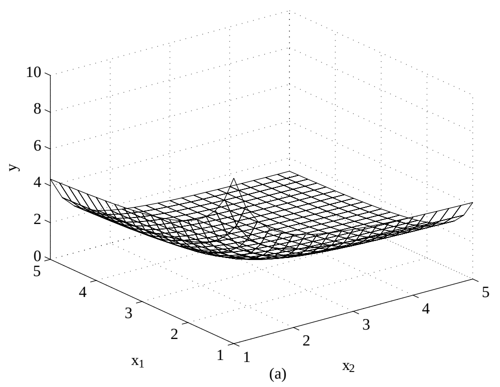
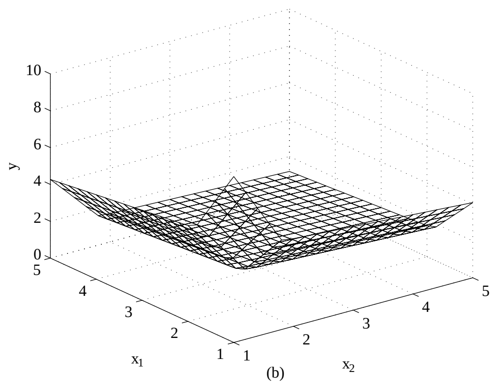


Figure7



(a)



(b)

Figure 8

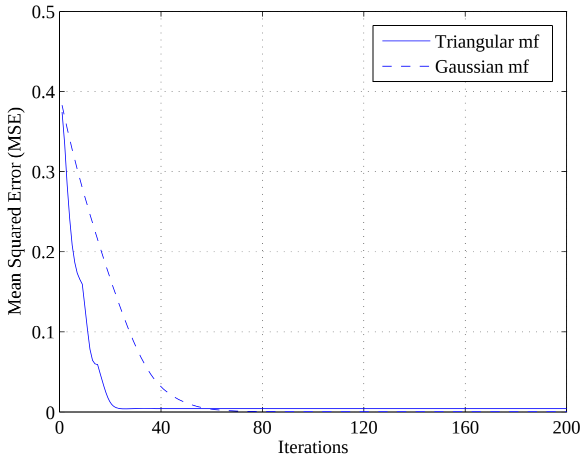


Figure 9

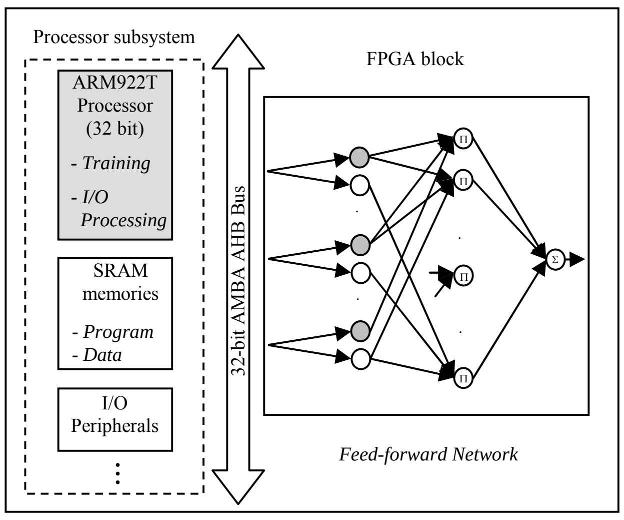


Figure 10

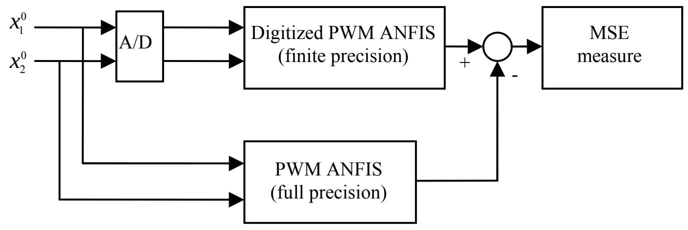

x_1^0
x_2^0
A/D
Digitized PWM ANFIS (finite precision)
+
-
MSE measure
PWM ANFIS (full precision)

Figure 11

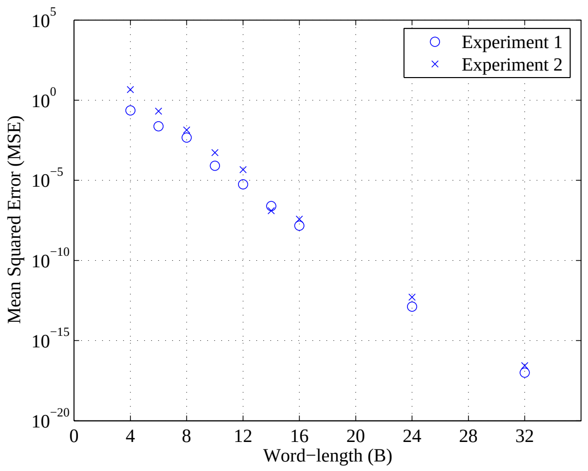


Figure 12

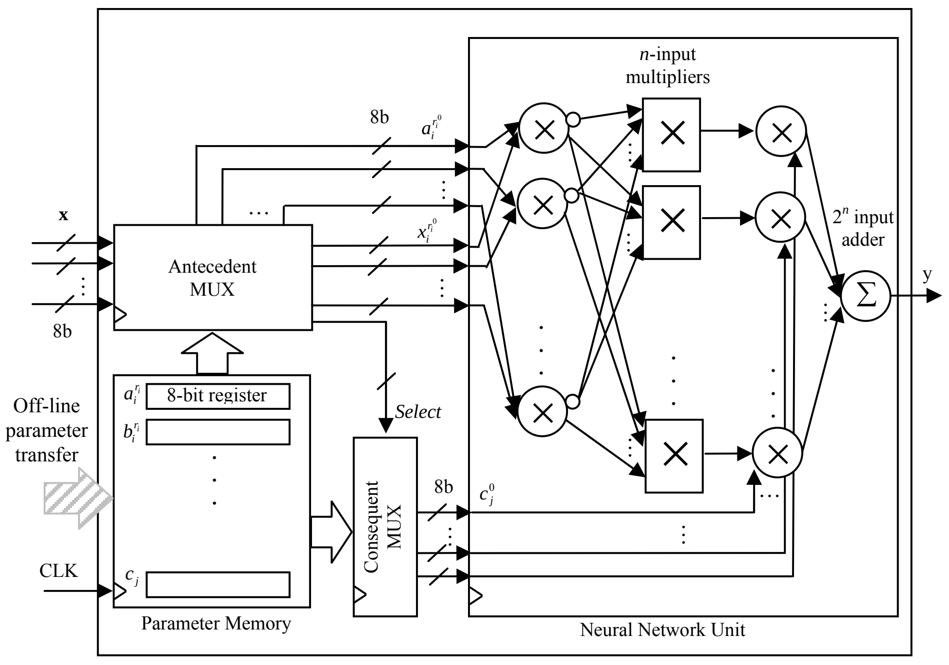


Figure 13

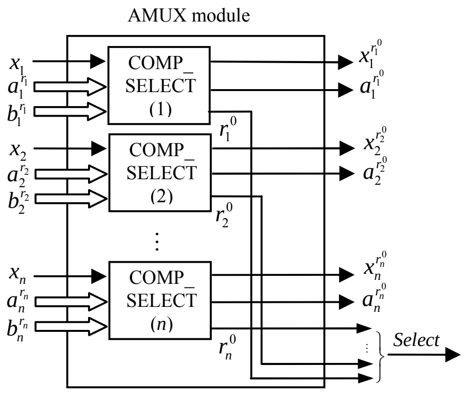


Figure 14

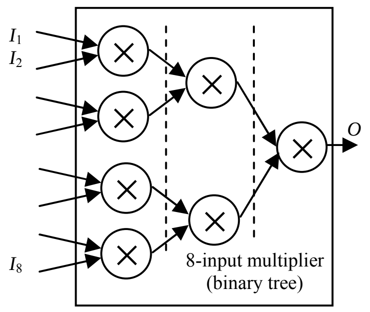


Figure 15

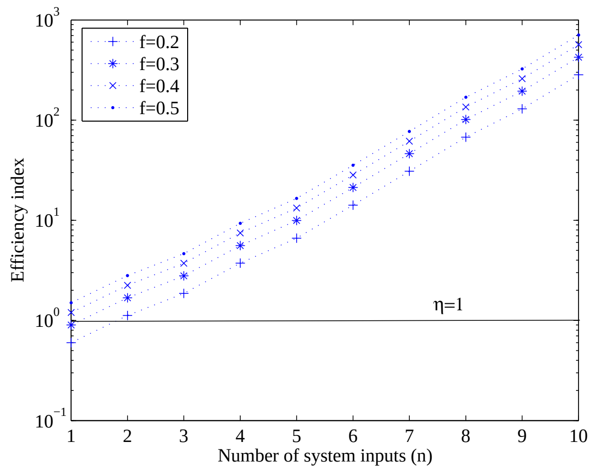


Figure 16

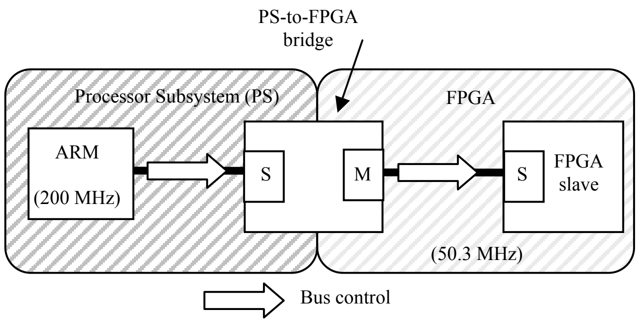


Figure 17

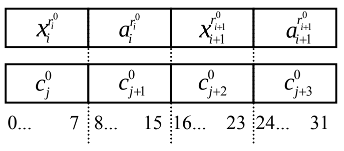


Figure 18

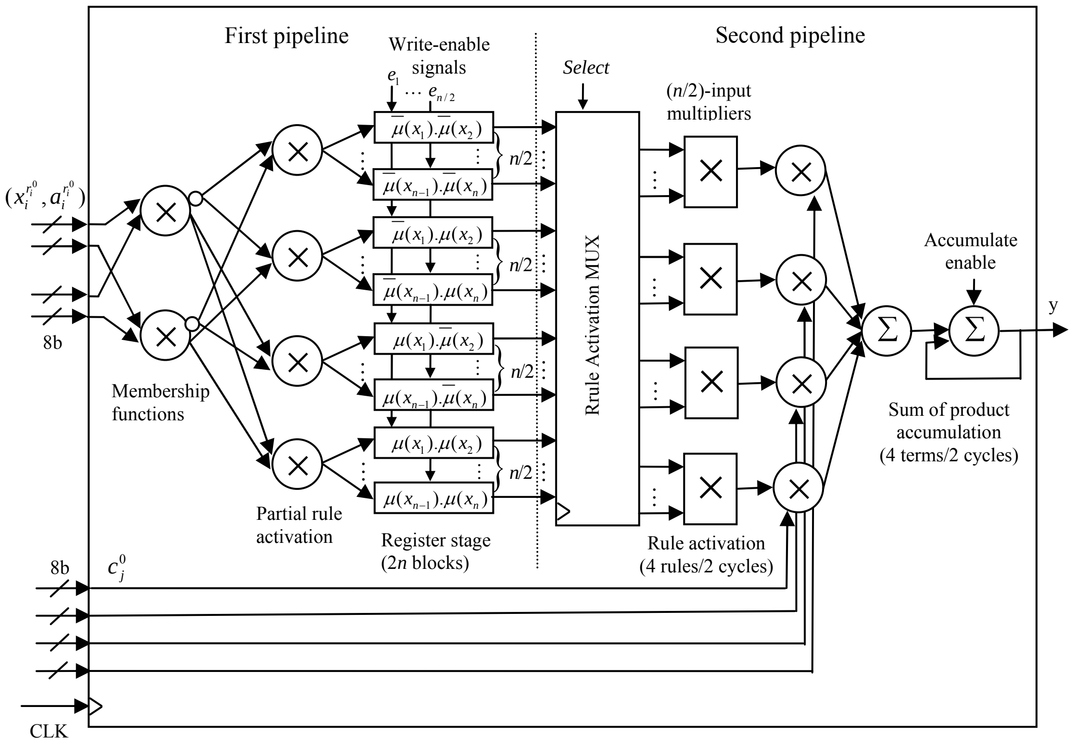
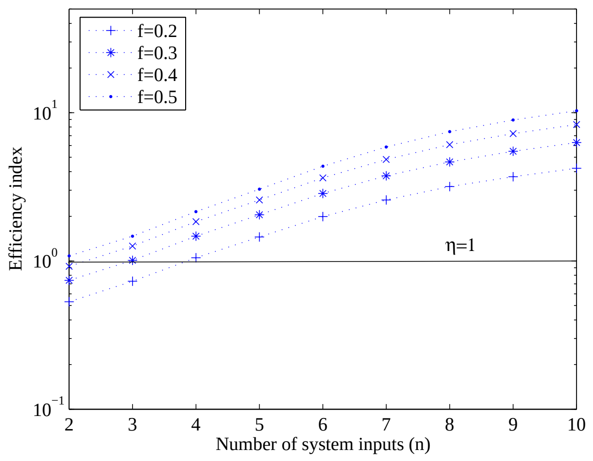


Figure 19



TABLES

TABLE I

COMPARISON ON THE GENERALIZED MEAN SQUARED ERROR (GMSE) OF EXPERIMENT 1 AND EXPERIMENT 2

NFS	Membership function type	Learning algorithm	GMSE Experiment 1	GMSE Experiment 2
Lin's System	Trapezoidal	GDM	0.0088	0.0537
Wong's System	Gaussian	GDM	0.0033	0.0472
Lee's System	Gaussian	Hybrid (LSE and GDM)	0.0023	0.0407
Our System	Triangular	Hybrid (LSE and GDM)	0.0039	0.0630

The NFSs above use 16 rules to approximate the function of experiment 1, while for experiment 2 our system uses 9 rules and the other three systems use 10 rules.

TABLE II

EVALUATION OF THE DATA TRANSFER RATE AND THE PARAMETER TRANSFER RATE FOR EACH LAYER OF THE PWM ANFIS

NFS	Layer 1	Layer 2	Layer 3
Input bits	$nB_I + 2nB_P$	$2nB_{O1}$	$2^n(B_{O2} + B_P)$
Output bits	$2nB_{O1}$	$2^n B_{O2}$	B_O
Total I/O bits	$n(B_I + 2B_P + 2B_{O1})$	$2nB_{O1} + 2^n B_{O2}$	$2^n(B_{O2} + B_P) + B_O$
Layer time	$\frac{2n}{P_1} n_1 T_{CLK}$	$\frac{2^n}{P_2} n_2 T_{CLK}$	$\frac{1}{P_3} n_3 T_{CLK}$
Data transfer rate (bits/s)	$\frac{(B_I + 2B_{O1})P_1}{2n_1 T_{CLK}}$	$\frac{(2nB_{O1} + 2^n B_{O2})P_2}{2^n n_2 T_{CLK}}$	$\frac{(2^n B_{O2} + B_O)P_3}{n_3 T_{CLK}}$
Parameter transfer rate (bits/s)	$\frac{B_P P_1}{n_1 T_{CLK}}$	--------	$\frac{2^n B_P P_3}{n_3 T_{CLK}}$

TABLE III

REQUIRED DATA TRANSFER RATE AND PARAMETER TRANSFER RATE FOR EACH LAYER OF A FULLY PARALLEL IMPLEMENTATION OF THE PWM ANFIS

NFS	Layer 1	Layer 2	Layer 3
Data transfer rate (bits/s)	$\frac{n(B_I + 2B_{O1})}{T_{CLK}}$	$\frac{2nB_{O1} + 2^n B_{O2}}{T_{CLK}}$	$\frac{(2^n B_{O2} + B_O)}{T_{CLK}}$
Parameter transfer rate (bits/s)	$\frac{2nB_P}{T_{CLK}}$	--------	$\frac{2^n B_P}{T_{CLK}}$

TABLE IV
COMPUTATION OF THE FEED-FORWARD PWM ANFIS BY USING A ONE CYCLE PER INSTRUCTION PROCESSOR

	SW clock cycles (n_{SW})
Preprocessing	$n(2NA-3)$
Layer 1	$3n$
Layer 2	$2^n(n-1)$
Layer 3	2^{n+1}
Total	$2^n(n+1)+2nNA$

TABLE V
COMPUTATION OF THE FEED-FORWARD PWM ANFIS BY MEANS OF THE FULLY PARALLEL ARCHITECTURE

	HW clock cycles (n'_{HW})
Preprocessing	1
Layer 1	1
Layer 2	$\log_2 n$
Layer 3	2
Total	$4+\log_2 n$

TABLE VI
COMPUTATION OF THE FEED-FORWARD PWM ANFIS BY MEANS OF THE PIPELINE ARCHITECTURE

	HW clock cycles (n'_{HW})	SW clock cycles (n'_{SW})
Preprocessing	--------	$n(2NA-3)$
Pipeline 1	n+3	--------
Pipeline 2	$1+\log_2 n+2^{n-1}$	--------
Total	$4+n+\log_2 n+2^{n-1}$	$n(2NA-3)$