Microprocessors and Microsystems xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

# An FPGA-based multiprocessor-architecture for intelligent environments

J. Echanobe [a], I. del Campo [a], K. Basterretxea [b], M.V. Martinez [a], Faiyaz Doctor [c]

[a] *Dept. of Electricity and Electronics, University of the Basque Country, Vizcaya, Spain*
[b] *Dept. of Electronics and Telecomm., University of the Basque Country, Vizcaya, Spain*
[c] *Dept. of Computing and the Digital Environments, Coventry University, Coventry, UK*

ARTICLE INFO

*Article history:*
Available online xxxx

*Keywords:*
Intelligent environments
NeuroFuzzy
FPGA
SoPC
Multiprocesssor

ABSTRACT

In this paper we propose a SoPC-based multiprocessor embedded system for controlling ambiental parameters in an Intelligent Inhabited Environment. The intelligent features are achieved by means of a Neuro-Fuzzy system which has the ability to learn from samples, reason and adapt itself to changes in the environment or in user preferences. In particular, a modified version of the well known ANFIS (Adaptive Neuro-Fuzzy Inference System) scheme is used, which allows the development of very efficient implementations. The architecture proposed here is based on two soft-core microprocessors: one microprocessor is dedicated to the learning and adaptive procedures, whereas the other is dedicated to the on-line response. This second microprocessor is endowed with 4 efficient ad hoc hardware modules intended to accelerate the neuro-fuzzy algorithms. The implementation has been carried out on a Xilinx Virtex-5 FPGA and obtained results show that a very high performance system is achieved.

## 1. Introduction

Ambient Intelligence (AmI) is a relatively new paradigm that proposes environments – e.g. public or private halls, rooms or spaces – capable of adapting to the preferences and needs of the people existing in them in order to make their daily activities easier and their daily life more comfortable [1–4] (AmI paradigm is also known as "Intelligent Inhabited Environments"). To reach this goal in its more general sense, the environments must be endowed with smart capabilities, such as recognizing the user and its situational context (i.e., context aware), being tailored to his/her needs, adapting itself in the changes of his/her preferences and interacting with him/her [5]. All these features are carried out by means of an amount of interconnected electronic devices (sensors, microprocessors, ad hoc hardware, actuators, communication support) distributed all around the environment, which, in addition, must be invisible to the user who only perceives the user interface.

The electronic devices for an AmI scenario must therefore be small, low-cost, with a low power consumption, and with a fast response, but also clever enough to cope with the above mentioned intelligent skills. However, this trade-off is hard to obtain: i.e., devices with intelligent capabilities demand high computational requirements which imply large and more expensive hardware and higher response times. Nevertheless, the increasing integration of electronic devices makes it possible at present to accommodate all the components of a typical electronic system on a single chip, commonly referred to as system-on-a-programmable chip (SoPC). Thus, it is possible to achieve high performance systems with a reduced size, cost and power consumption [7,6].

In this paper we propose a FPGA-based SoPC for controlling a number of ambiental parameters in an Intelligent Environment. The intelligent capabilities of the system are achieved by means of Neuro-Fuzzy Systems (NFS) which have the ability to learn and adapt from incoming samples (the main feature of neural networks) and also the ability to perform imprecise or vague reasoning (the main feature of fuzzy systems) [8]. This twofold feature makes NFS very suitable for AmI environments, as has been shown by the authors in previous works [10–12].

Bearing in mind this NFS structure, an efficient architecture has been developed to be implemented in the FPGA. The proposed architecture is based on 2 microprocessors (soft-cores) and ad hoc designed hardware modules acting as high performance coprocessors to accelerate the neuro-fuzzy algorithms. The training and/or adapting procedure is addressed by one microprocessor, whereas the reasoning (i.e. the on-line operating mode) is performed by the other.

As will be shown in the paper, the proposed architecture provides important advantages with respect to other existing AmI implementations. These advantages are related to issues like

http://dx.doi.org/10.1016/j.micpro.2014.07.005
0141-9331/© 2014 Elsevier B.V. All rights reserved.

efficiency, cost and size, among others. In addition, to validate our proposal, the system has been tested with a data set collected at the intelligent dormitory (iDorm) which is a real ubiquitous computing test bed from Essex University [13].

The rest of the paper is organized as follows: Related works are presented in Section 2. Taking into account the existing works, we also explain our proposal. Section 3 presents the NFS used here which in fact is a modified version of the well known ANFIS model that has been called PWM-ANFIS by the authors because of its Piece-Wise Multilinear (PWM) behavior [11]. Both the theoretical aspects and the modeling performance are analyzed. Particular learning and modeling examples from real AmI experiments are also addressed. In Section 4 the proposed FPGA-based multiprocessor architecture is described in detail. Also the benefits of using 2 processors in this particular AmI system are discussed. Section 5 presents the particular implementation details: the FPGA used, the performance achieved, the device resources utilization and the execution times. Finally the main conclusions of the work as well as future research are explained in Section 6.

## 2. Related works and proposed system

### 2.1. Ambient intelligence

The AmI paradigm has existed for little more than a decade. It is however a very active field, as is shown by the large number of publications that can be found in the literature. For example, there are international journals dedicated almost exclusively to this subject [14,15], and specialized books [16–19] that analyze carefully most of the AmI aspects. Also, some of the most important journals have published special issues on this particular area [20–22]. In addition, different monographic conferences take place every year [23,24] with a large number of contributions. Among all existing works, we could point out, for example, [13,25,26] as those of the most cited by other authors. More recent works which also address the AmI paradigm in an exhaustive way are [27,28].

However, in most existing works, the aim of the research is to design and validate different methodologies and algorithms rather than propose a particular hardware implementation. Even when these works present any kind of physical implementation [29–32] the authors hardly touch on aspects such as power efficiency, size and cost of the implementations. Some of the very few works where the efficiency of the implementation is addressed are for example [31,33,34]. Such proposals are limited, however, to the set of sensors or to the wireless communication issues among the different nodes, but do not deal with the intelligent hardware counterpart (i.e., intelligent agent).

### 2.2. Computational intelligence

Computational Intelligence (CI in the following), also called Soft Computing [35], is a discipline that comprises a number of bioinspired techniques or tools for dealing with problems which are very hard to solve, due to complexity, non-linearities, high dimensionality, absence of an analytical model, etc. CI techniques have in general the capacity of handling information that is imprecise, vague, incomplete or ambiguous. The most representative paradigms are fuzzy systems – able to perform imprecise reasoning-[9], neural networks – which can learn from a set of samples – [36] and the genetic algorithms – which make a search in the space of solutions – [37]. Furthermore, many systems combine two or more of these techniques such as, for example, neuro-fuzzy systems [8,38] or fuzzy-genetic systems [39]. CI techniques have been used successfully in problems where ordinary tools fail to provide a solution. One important inconvenience of these methodologies is, however, that they require a large amount of computational resources. In consequence, it becomes very difficult to carry out high performance systems, specially if they have to be small and low powered, as is the case in embedded systems. This drawback makes it necessary, in many cases, to design hardware accelerators to cope with the timing requirements.

Regarding the intelligent abilities required in an ambient intelligence scenario, there are several authors who have proposed CI algorithms to cope with the problem. For example, fuzzy systems have been proposed in the (above mentioned) intelligent dormitory (iDorm) [13] and also in the fuzzy agent of an intelligent classroom (iClass) [52] which is a testbed for educational ambient intelligence systems. In [53] a Neuro-Fuzzy system is proposed for the management of comfort and energy conservation. Another meaningful example is proposed in [54] where Fuzzy Systems and Genetic Algorithms are combined to develop an intelligent agent in commercial buildings.

### 2.3. FPGA implementations of CI algorithms

Many researchers have proposed FPGA-based implementations of CI algorithms when high performance is required in the targeted applications. The advantages of FPGAs are mainly due to the high parallelism that can be achieved for performing computations. Hence, these solutions broadly overcome the computational power of a microprocessor-based software solution. We can find a lot of works in the literature about FPGA implementations of different CI algorithms. Among the most recent and relevant papers we can find FPGA implementations of fuzzy systems [42,41,43,47], neural networks [44–46], Genetic algorithms [48–51] or Swarm Intelligence [40]. In these works, the applications comprise different areas such as Robotics, control of real plants (DC motor, boost converter), electrical power systems or speech recognition. In all the cases, the viability of the proposal is shown, as well as the advantages of such an implementation particularly in terms of velocity, size and power efficiency. In this sense, our group has proposed in [10,11] an FPGA-based embedded system for an AmI scenario in which an NFS system with learning/adapting and reasoning features is integrated in a single device. The aim of those proposals was to achieve an electronic system that was small and efficient, but also smart enough to cope with the requirements of an AmI scenario. One drawback of that proposed system is however, that it is forced to stop its on-line activity (i.e. interaction with the environment) each time it needs to adapt its knowledge (due mainly to changes in the environment or in the user preferences). This can be a serious problem in some environments if the system is acting in a particular situation (i.e: context) in which it should not be interrupted (e.g., assisting a user to perform a task or regulating a delicate device).

### 2.4. Proposed system

In this paper we propose a FPGA-based system on chip (SoC) with intelligent abilities intended for an AmI scenario. The proposal is based on a Neuro-Fuzzy system as intelligent agent.The novelty of this proposal is a new efficient architecture in which the adapting (i.e. learning) and reasoning processes are executed in different microprocessors as independent tasks which only communicate by means of several flags. Although this new design is proposed to avoid the pitfalls encountered by the group previously (as explained above), there are also some other additional advantages which make the proposal yet more valid. For example, independent tasks imply that designing and updating the software is easier, more flexible and more reliable. Also, the architecture of every microprocessor can be customized for its particular task, thus increasing performance and efficiency.

## 3. Neuro-fuzzy system

### 3.1. PWM-ANFIS

As we have stated above, the intelligent behavior of the proposed AmI system is addressed by means of an ANFIS-like system. An ANFIS system [55] is a Fuzzy Inference System whose parameters – membership functions and consequents – are trained by means of Neural Network algorithms. In fact, such a system can be viewed as a particular Neural Network that is functionally equivalent to a Fuzzy Inference System. Hence, it exhibits both the linguistic knowledge representation of fuzzy systems and the learning abilities of Neural Networks. To understand the structure of an ANFIS system we must pay attention first to the fuzzy system it represents.

Consider a rule based $n$-input fuzzy system with $m$ antecedent functions per dimension. In general, we have a total of $m^n$ rules where the $j$th rule can be expressed as:

$$R_j: \quad \text{IF } x_1 \text{ is } M^1_{j_1} \text{ and } \ldots \text{ and } x_n \text{ is } M^n_{j_n} \text{ THEN } y \text{ is } c_{j_1 j_2 \cdots j_n}$$

where $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}), \mathbf{x} \in \mathbf{R^n}$ is the input vector, $M^1_{j_1}, \ldots M^n_{j_n}$ are linguistic labels associated with the membership functions $\mu_{ij_i}(x_i)$, $1 \leqslant i \leqslant n$; $1 \leqslant j_i \leqslant m$ (i.e., antecedents of the rules), $y$ is the output variable and $c_{j_1 j_2 \cdots j_n} \in R$ is a crisp consequent (i.e. a singleton).

If the center of gravity defuzzification method is adopted, the output of the system is given by (1)

$$y = \frac{\sum_{j=1}^{m^n} w_j c_j}{\sum_{j=1}^{m^n} w_j} \tag{1}$$

where

$$w_j = M^1_{j_1}(x_1) \cdot M^2_{j_2}(x_2) \cdot \ldots \cdot M^n_{j_n}(x_n). \tag{2}$$

The computation of the output of this system – Section 3 – for a given input (i.e., the inference mechanism) can also be performed by a Neural Network like the one shown in Fig. 1.

– The first layer computes the membership functions:

$$M^i_{j_i}(x_i) = \mu_{ij_i}(x_i) \tag{3}$$

– The second layer computes the values $w_j$ $(1 \leqslant j \leqslant m^n)$ as the product

$$w_j = M^1_{j_1}(x_1) \cdot M^2_{j_2}(x_2) \cdot \ldots \cdot M^n_{j_n}(x_n) \tag{4}$$

These values are also called the activation of the rules.

– The third layer normalizes these values by dividing each one by the sum of all of them:

$$\overline{w}_j = w_j \bigg/ \sum_{i=1}^{m^n} w_i \tag{5}$$

–The fourth layer multiplies each term by the consequent:

$$\overline{w}_j c_j \qquad (1 \leqslant j \leqslant m^n) \tag{6}$$

–Finally the output of the network is provided by the fifth layer which aggregates the overall output as the summation

$$y = \sum \overline{w}_j c_j \tag{7}$$

As can be shown, this network performs the same function as the previous fuzzy system.

To train the above network, a hybrid algorithm has been proposed [55]. The algorithm is composed of a Least Squares Estimator (LSE) process, followed by a back propagation (BP) algorithm. LSE computes the consequents and BP adjusts the parameters of the antecedents. Although the BP algorithm could train the network alone, the training process can be accelerated with the LSE. This is possible due to the fact that the output of the network is linear in the consequent parameters. This hybrid algorithm is executed iteratively from a collection of training data (i.e., input/output pairs) until all the parameters are adjusted. Later, it can be executed again whenever one or more new training pairs are provided (for example, due to a change in the users' preferences in the case of AmI environments). This is known as adaptive learning or online training. In this situation the parameters can be updated with fewer iterations.

Due to the huge amount of information and the large number of parameters that are involved in an Intelligent Environment, we have used a modified model called PWM-ANFIS (PieceWise Multilinear ANFIS) which has been proposed [10,11] to reduce the computational requirements in high dimensionality systems.

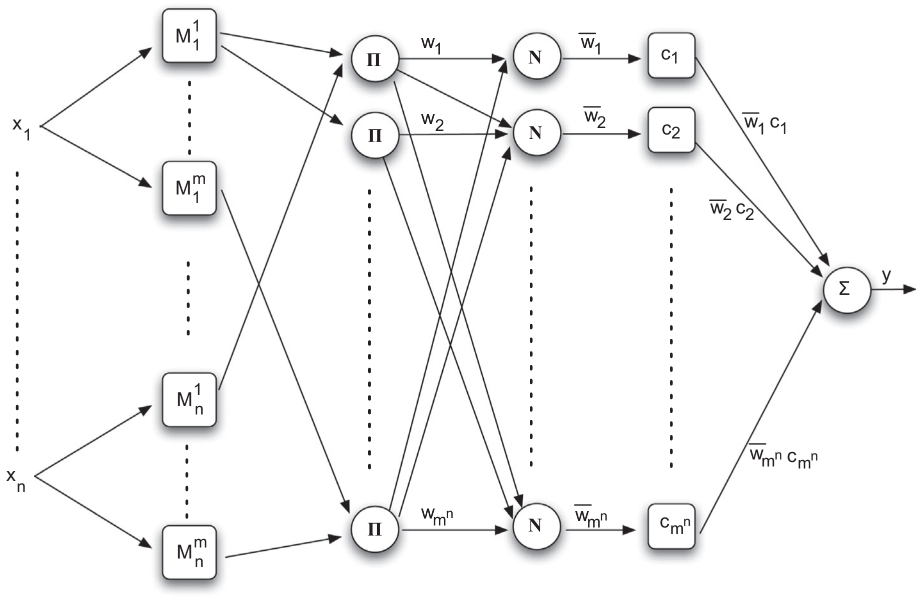


**Fig. 1.** ANFIS network structure.

In particular, we have shown how, by introducing some restrictions on the membership functions (i.e., antecedents), a much more simplified system can be obtained with hardly any loss of the learning and approximation capabilities. These restrictions basically involve the use of normalized triangular membership functions overlapped by pairs as is shown in Fig. 2. As a result, only two antecedents per input have non zero values for every new incoming vector and hence, only $2^n$ rules become active each time (instead of $m^n$ in the unrestricted ANFIS). Also, the divisor in Eq. (1) becomes now 1 and therefore the division is no longer necessary. The computation of the output is now given by

$$f = \sum_{j=1}^{2^n} w_j c_j \tag{8}$$

To perform an inference we have now to sum $2^n$ terms $w_j c_j$ where each term $w_j$ is computed by multiplying $n$ membership functions (Eq. (4)). As the antecedents are now triangles, every membership function evaluation is also simplified because it is obtained by solely multiplying the slope of the triangle $m^i_{j_i}$ by the offset of the input with respect to the triangle's left corner $a^i_{j_i}$.

$$M^i_{j_i}(x_i) = \mu_{ij_i}(x_i) = m^i_{j_i}(x_i - a_i) \tag{9}$$

In addition, because the membership functions are normalized, the two active antecedents have complementary values and therefore the computation of one of them gives directly the other one simply by performing a logical complement.

$$M^i_{j_i+1}(x_i) = \mu_{ij_i+1}(x_i) = 1 - M^i_{j_i}(x_i) \tag{10}$$

As we can see, the inference process is computed by means of only a few sums and products so it is greatly simplified. Also the training procedure is simplified because the parameters of the antecedents that have to be adjusted are only the init points or offsets ($a^i_{j_i}$) of every triangle (i.e., note that the slope can be obtained from these points thanks to the imposed restrictions).

## 3.2. Modeling performance

To test the modeling ability in dealing with AmI environments, the proposed PWM-ANFIS system has been trained with a data subset from a real experiment. The data have been collected at the intelligent dormitory (iDorm) which is a real ubiquitous computing test bed from Essex University [13]. In this experiment, a normal room is provided with a large number of interconnected sensors, actuators and processing elements. Inside this room a user spent several days, performing ordinary tasks and interacting with an AmI system which gradually learns and adapts to his behavior.

The data being used here relate to seven input sensors and to four controlled actuators. The input sensors are as follows: internal light level, external light level, internal temperature, external temperature, chair pressure, bed pressure and time. The actuators are four variable-intensity spot lights. This means that we have to deal with four PWM-ANFIS, each one with seven input variables. In order to reduce the complexity of the problem and hence to obtain a more efficient system, a previous dimensionality reduction has been explored. This reduction, which has been carefully carried out in order to preserve as much as possible the modeling performance, has led to an input space of just four variables for every PWM-ANFIS. The obtained dependence between outputs and inputs is depicted in Table 1.

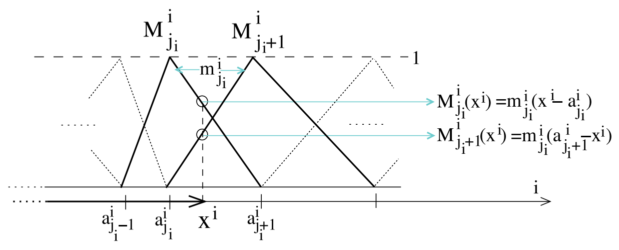
The next step was to train every PWM-ANFIS with the data collection. Around 400 input/output data pairs for every output variable are available in the collection provided. Each one of these data sets was randomized 6 times and split into a training set and a validation/test set consisting of around 270 and 130 instances respectively. Yet, to avoid the overfitting effect that neural systems trained with large amounts of experimental data usually show, we used the subset comprised of 130 instances as a validation set and not as a test set. The validation set was then monitored during the training phase in order to set the final PWM-ANFIS with the parameters obtained for the minimum root mean squared error (RMSE) on the validation set and not for the minimum RMSE on the training set (early stopping). In addition, different numbers of antecedents and different learning rates (i.e., a parameter of the BP algorithm) were also tested in the experiments. Finally, the experiments were repeated with generic ANFIS systems to compare the modeling ability.

After exhaustive experimentation, we can conclude that the system shows quite good learning and approximation capabilities. In addition, the results are not far from those obtained with the ANFIS model without restrictions. Table 2 shows the average RMSE obtained for each one of the 4 PWM-ANFIS and also the respective values obtained with a generic ANFIS. As we can see, the error values obtained with the PWM-ANFIS systems are very small and comparable (i.e., the same magnitude order) with those obtained with the generic ANFIS systems.

To illustrate the learning process we also show in Fig. 3 the evolution of the RMSE obtained during the training procedure for the four PWM-ANFIS systems in some of the multiple experiments. As we can see, after about 1000 iterations, the system is trained and small error values are reached.

## 4. A multiprocessor architecture

In this section an efficient architecture is presented for the iDorm experiment described above. To be exact, we have developed an architecture based on two microprocessors and four high performance hardware cores (one for each PWM-ANFIS module), to control the intensity of four spot lights. The architecture also contains memory blocks, buses and I/O peripherals. Thus, a complete SoPC is obtained. Being able to implement the entire system into devices as small as FPGAs, makes it much more feasible to carry out an intelligent environment where, as indicated above, the devices to be deployed must be small and with a low power and cost.



$M^i_{j_i}$
$M^i_{j_i+1}$
$m^i_{j_i}$
1
$M^i_{j_i}(x^i) = m^i_{j_i}(x^i - a^i_{j_i})$
$M^i_{j_i+1}(x^i) = m^i_{j_i}(a^i_{j_i+1} - x^i)$
i
$a^i_{j_i-1}$
$a^i_{j_i}$
$x^i$
$a^i_{j_i+1}$

**Fig. 2.** The 2 active antecedents in the $i$th input, for an incoming value $x_i$. $m^i_{j_i}$ and $a^i_{j_i}$ denote respectively the slope and the left corner of the triangles.

**Table 1**
Dependence between input and output environment parameters in the 4 PWM-ANFIS subsystems.

| | Light 1 | Light 2 | Light 3 | Light 4 |
|---|---|---|---|---|
| Int. Light | | | | |
| Ext. Light | X | X | X | X |
| Int. Temp. | | | | |
| Ext. Temp. | | | | X |
| Chair Press. | X | X | X | X |
| Bed Press. | X | X | X | X |
| Hour | X | X | X | |

**Table 2**
Average root mean squared error (RMSE) obtained in the 4 PWM-ANFIS.

| | Light 1 | Light 2 | Light 3 | Light 4 |
|---|---|---|---|---|
| PWM-ANFIS | 0.0787 | 0.0785 | 0.0515 | 0.0502 |
| ANFIS | 0.0683 | 0.0658 | 0.0376 | 0.0414 |

Multiprocessor architecture is a common issue in high performance systems. In fact, one of the most evident reasons for a system to be designed using a multiprocessor architecture is the performance requirement: e.g., two processors acting in parallel can execute a task twice as fast as only one. There are, however, some other important reasons that also have to be taken into account. One of them is the presence of tasks with very different execution time schemes: i.e one processor is intended for computing and providing a real-time output whereas another one can be in the background performing a non-critical task. Another reason arises in a scenario where different-nature algorithms are present, that is, one function is based on repetitive and simple operations, whereas another one implies very irregular and conditional execution. In this case, it is worth using 2 different processors whose architectures are respectively oriented to each algorithm. Some other reasons for using several processors can be also mentioned, such as reliability, redundancy or data streaming in a pipe-line fashion.

The ANFIS-like system presented in this work – intended for an AmI scenario – is an example of a system demanding a multiprocessor architecture. On the one hand, we have that the inference process must be performed in real-time; that is, the system must interact permanently with the environment providing new outputs all the time. On the other hand, from time to time, the system must learn from incoming new samples and hence adapt its parameters to the changing user preferences. However, this is a process that can be performed in the background without rigid timing constraints. These two tasks are clearly two different functions, each one with particular time requirements. Also, the inference process is in fact the output of the neural network, and as such, is computed by similar and repetitive operations (i.e., the computations of interconnected layers of neurons). The learning process, is however, a very irregular algorithm that contains different functions including matrix inversions and it requires high precision in the computations.

Taking into account the above issues, we have developed an architecture based on two microprocessors together with 4 ad hoc hardware modules (called PWM-Cores) to accelerate the neural-network computations. Also, several buses, internal memories (on-chip RAM blocks) and I/O peripherals are present in the architecture. In particular, each microprocessor is a Xilinx soft-core called "MicroBlaze" which is a 32-bit RISC Harvard architecture soft processor core with a rich instruction set optimized for embedded applications. Also, it can be tuned by the designer to better fit the requirements of the application to which it is dedicated. The proposed architecture is shown in Fig. 4 and is explained in detail in the following.

### 4.1. Microprocessor 1

The first MicroBlaze (MB1) (the left one in Fig. 4) is intended for the training process and hence is implemented with a Floating

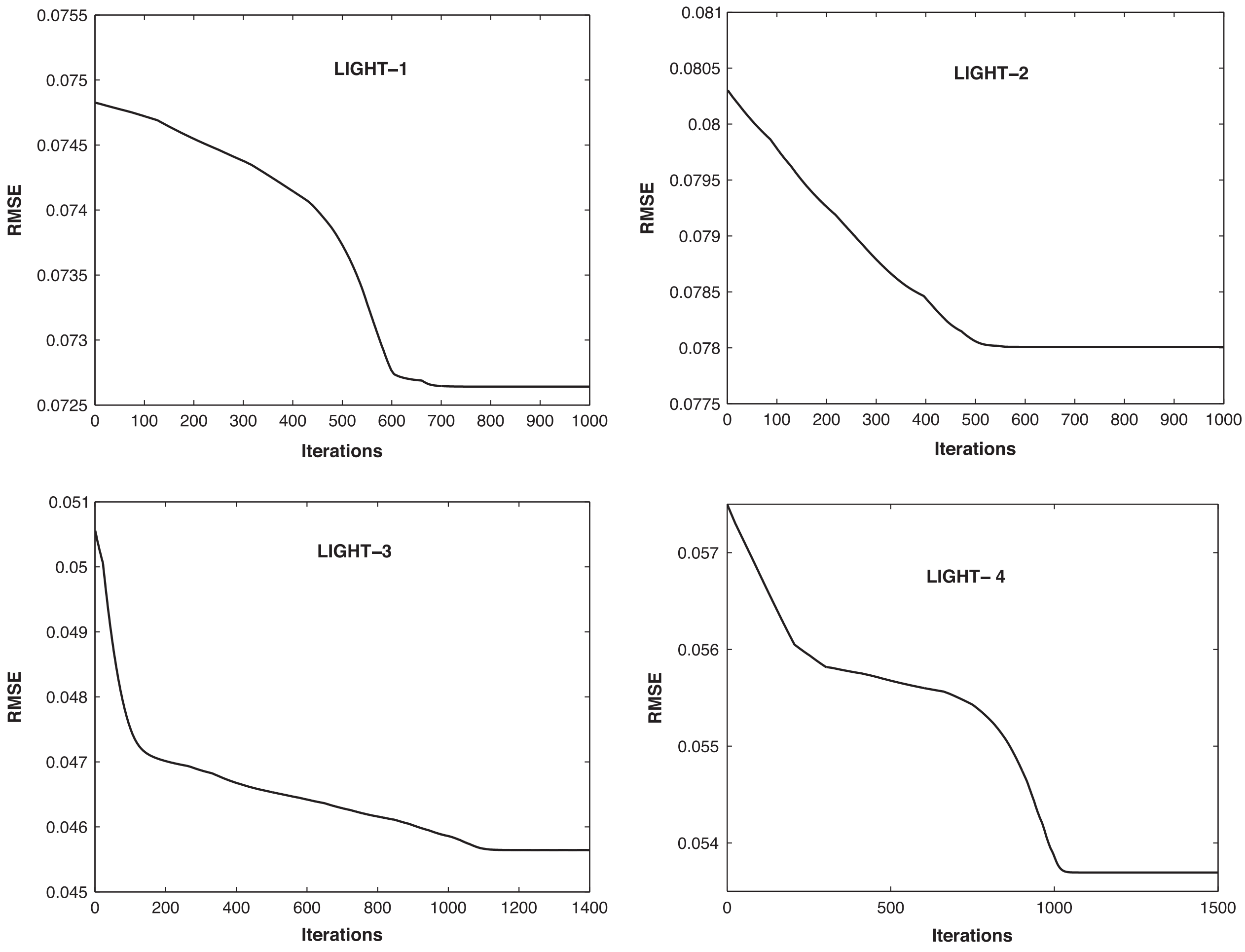


**Fig. 3.** Example of the RMSE evolution during a PWM-ANFIS training for light 1, light 2, light 3 and light 4 respectively.

Point Unit (FPU) and also with some other hardware features like an integer multiplier, a barrel shifter or memory caches in order to address the required precision and intensive calculations. This microprocessor takes as input the training data collections, which are stored in an external memory, and performs the hybrid training algorithm explained in Section 2. Actually, it performs the training algorithm four times: one for each PWM-ANFIS system. After the training process, MB1 stores the results of the algorithm, the offsets of the triangles and the consequents – in a RAM memory block that is shared by the second MicroBlaze (MB2). Also, it stores a flag (t-flag) to indicate that trained parameters are available. The training algorithm is carried out once at the beginning of the whole process and again whenever a new training pair is received (i.e. online-training). This last case is notified by MB2 who also stores a flag (ol-flag) and provides the new data (both stored in the shared memory). Therefore, when MB1 finishes the initial learning, it waits until this flag is set, in which case it loads the new data and performs on-line learning (see Section 2).

The program to perform the training algorithm is stored in a RAM memory block whose size is adjusted – in the design phase – to the size of the code. This memory block, and also the shared memory block are connected to the MicroBlaze with a dedicated memory bus from Xilinx called LMB (local memory bus). The LMB is a fast, local bus for connecting the MicroBlaze processor instruction and data ports to high-speed peripherals, primarily on-chip block RAM (BRAM).

## 4.2. Microprocessor 2

MB2 deals with the following tasks: (1) control the global system operation; (2) handle input/output data; (3) provide on-line data to MB1; (4) write/read inference data to/from PWM-Cores (described below). Hence, MB2 does not need to execute complex algorithms and is therefore implemented without extra features. For this reason, it needs much fewer resources of the FPGA than MB1. On the other hand, MB2 is connected – like MB1 – to two RAM memory blocks through another LMB bus. One of the blocks is shared with MB1 to access the trained parameters (so it can configure the PWM-Cores) and to store the on-line data, and the other block contains the program to be executed.

To perform the I/O processing, MB2 first takes the five input values (see Table 1) which are provided by different sensors. Then, it performs a data conditioning so they can be better processed; e.g., scaling, normalization and truncating to a fixed length. In particular, every input value is fixed to a 8-bit length, and then, the four values for each system are packed into a single 32-bit word. This packing allows the four inputs to be sent to every core in just a single transaction. The cores receive these inputs, perform the inferences and send back the results to the micro which sends them out. Note however, that prior to performing any inference, MB2 has to load the trained parameters from the shared memory and send them to the cores so they can configure the algorithms.

MB2 also checks continuously whether a new training pair is provided. This occurs for example when the user manually modifies any output (i.e., the light levels). In this case MB2 stores in the shared memory this modified value, the related four actual inputs and also a flag (ol-flag) so MB1 can perform an on-line training.

## 4.3. PWM Cores

These 4 PWM-Cores – one for each PWM-ANFIS – are ad hoc hardware cores that actually perform the system inferences. They have been designed carefully so they can perform the operations involved in the PWM-ANFIS outputs calculations in a very efficient way.

Fig. 5 shows the internal architecture of the developed PWM-Cores. The design of these cores has been carried out by exploiting the parallel nature of the FPGA resources. In this way, an efficient and high performance architecture has been achieved. Each one of these cores is connected to the MB2 by means of the Fast Simplex Link (FSL) bus from Xilinx, which is a uni-directional point-to-point communication channel bus used to perform fast communication. This bus is configured as a 32-bit wide bus and hence, just a single transference is required to receive the 4-input data (i.e., 8-bit data) from the MicroBlaze. The core is composed of the following three main modules or units.

First, we have a local RAM block to store the trained parameters needed for executing the algorithms. To accelerate the computations, not only the consequents and the offsets of the triangles

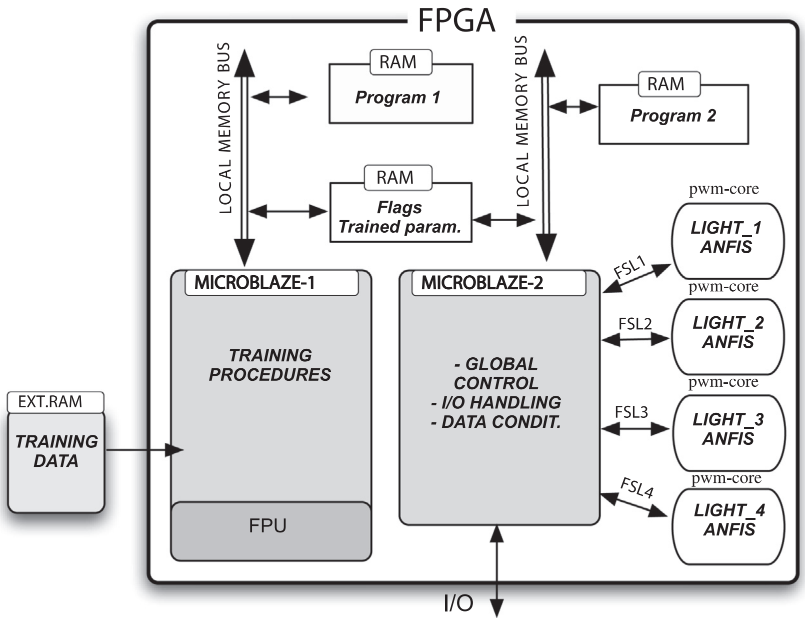


**Fig. 4.** Architecture of the global system.

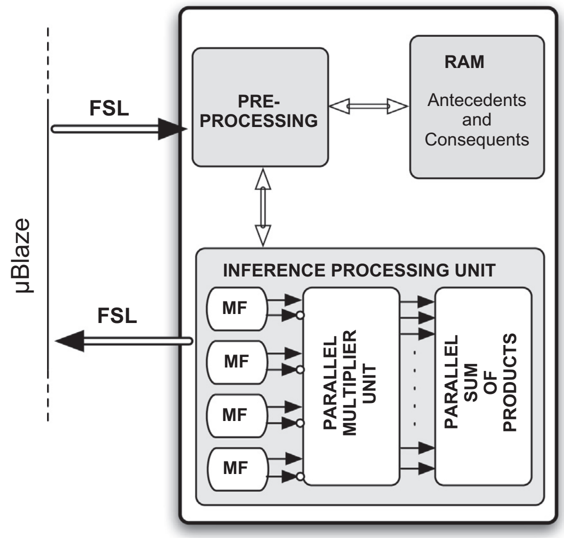


**Fig. 5.** Internal architecture of the PWM-Cores.

are stored but also the slopes of the triangles, which are previously computed by MB2.

Second, the module called "Preprocessing unit" is designed to receive the four input data from MB2 and to determine which are the respective 16 active rules ($2^4$). As a result, it loads the involved parameters from the local memory (slopes and consequents) and also computes the input offsets.

The third main module is called the "Inference Processing Unit". It takes the involved offsets, slopes and the consequents together with the loaded parameters and computes Eq. (8). It has three parts or stages.

–First of all, the "MF" units compute the membership values by multiplying the slopes with the respective offsets (Eq. (9)). Although 8 membership values must be computed (2 non-zero antecedents per dimension), only four "MF" modules are needed because, as we have explained in Section 2, the two membership values in every dimension have complementary values (Eq. (10)). Therefore, we have only to include a logical complement action in every module. This is indicated in Fig. 5 where two complementary outputs in the modules have been drawn in.
– The next step is carried out by the "Parallel Multiplier Unit" which computes the 16 activation values $w_j$ by multiplying in each case four membership values (Eq. (4)).
– Finally, the last unit multiplies every $w_j$ by the related consequent and sums the 16 terms, thus providing the output of the algorithm (Eq. (8)). This output is sent back to the microblaze as a 32-bit word.

As a result of the high parallelism achieved in the design, each core is able to perform an inference in just 10 clock cycles plus 2 extra cycles for receiving the inputs and sending the result.

### 4.4. Global functionality

Let us explain with the aid of Fig. 6 the operation of the whole system. To better understand how the whole system works, let us assume that only one PWM-Core system is present. The operation of the complete system (i.e., with 4 PWM-Cores) can be easily explained by adding for every PWM-Core – in a sequential way – the steps here showed.

When the system boots, both MB1 and MB2 start to execute their respective programs. MB1 takes the training data set stored in an external memory, and performs the learning algorithm. When it finishes, it stores the trained parameters in the shared memory and notifies MB2 by setting the t-flag. Then, the microprocessor waits until ol-flag is 1, which means that one or more new training data are provided by MB2. In this case, MB1 takes the new data, executes an on-line training and stores the updated parameters and also the flags.

MB2 starts its program and waits until MB1 finishes the learning process (t-flag = 1). Next, it loads the trained parameters from the shared memory and sends them to the respective PWM-Cores. It then proceeds with the inferences cycle, as described above (i.e. sending and receiving data to/from PWM-Cores). If new training I/O pair is provided (as explained before), it stores the pair and sets the ol-flag to indicate MB2 that a new training pair is available. MB2 remains in this cycle (real-time operation) until MB1 indicates that trained parameters have been updated due to a on-line learning. If this happens, MB2 stops the inferences, loads the updated parameters from the shared memory and sends them again to the PWM-Cores. Finally, it resumes the inferences.

## 5. Implementation results and discussion

The FPGA used to implement the proposed embedded system is the XC5VSX50T device of the Xilinx Virtex 5 family [56]. We selected this family because it is optimized for memory-intensive applications and digital signal processing (DSP) and hence, it contains a large number of memory blocks and multiplier units (i.e., DSP blocks). Nevertheless, this device is one of the smallest of this family (the second of four available sizes). It has 8.160 Slices (each Slice contains four 6-input LUTs and four flip-flops), 288 DSPs (each DSP consists of a multiplier, an adder, and an accumulator), 132 RAM memory blocks of 36 Kbits each, and 6 PLLs.

The development of the whole system has been carried out using different tools and strategies. The PWM-Cores have been designed in VHDL language – to optimize their design – and then synthesized with the "ISE Design Tool" 12.4 from Xilinx [57]. Previously, the "ModelSim" 6.4 environment [58] has been used to verify the functionality of the design. We have used the EDK software (which belongs to ISE suite) to instantiate the different Xilinx IP modules (MicroBlazes, buses, memory blocks, ...) and to assemble the whole system. The software has been developed first with MATLAB, then with the "GNU development tools" and finally integrated into the system with EDK.

*FPGA resources:* After hardware synthesis, EDK reports the following FPGA resource usage: 13464 Slices flip-flops (41%), 17106 Slices LUTs (52%), 239 DSPs (83%) and 66 RAM Blocks (50%). As can be seen, although the system is rather complex, the FPGA is not fully used and therefore, a smaller – and also cheaper – device could be used. Table 3 shows the resources required by each functional module. We can also see that the modules demanding more resources are the PWM-Cores. This is due to the fact that these modules have been designed with a full parallel architecture so they can provide a very high speed response. The different amount of resources needed by the two microblazes can be also seen in the table. Microblaze 1 requires many more resources than Microblaze 2 because, as explained above, it has been endowed with a floating point unit and additional hardware elements to perform training algorithms with sufficient precision and velocity. In contrast, Microblaze 2 does not perform floating point operations so it requires fewer resources. Another module which demands a considerable amount of resources is the external RAM controller. We cannot leave it out because the training algorithms need a large amount of data to train the ANFIS systems, and the internal memory is not large enough to cope with them.

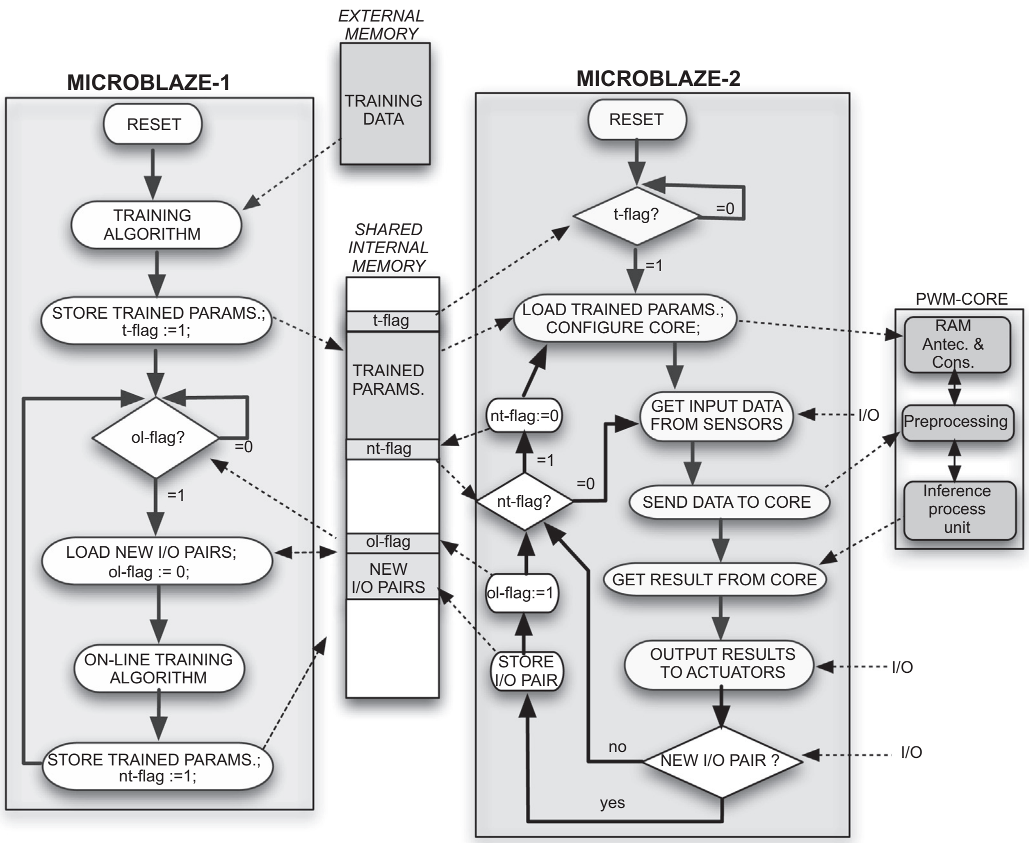


**Fig. 6.** Global functionality.

**Table 3**
FPGA resources utilization.

| | FFs | LUTs | BRAM | DSPs |
|---|---|---|---|---|
| $\mu$B1 | 2433 (7%) | 3008 (9%) | 21 (16%) | 6 (2%) |
| $\mu$B2 | 1487 (5%) | 1569 (5%) | 0 | 3 (1%) |
| 4 PWM-Cores | 5316 (16%) | 8952 (27%) | 0 | 228 (79%) |
| Ext. RAM Contr. | 3338 (10%) | 2198 (7%) | 7(5%) | 2 (0.7%) |
| Internal RAM. | 0 | 0 | 38(29%) | 0 |
| Buses | 546 (1.7%) | 1058 (3%) | 0 | 0 |
| I/O | 275 (0.8%) | 267 (0.8%) | 0 | 0 |
| Clk, Reset | 69 (0.2%) | 54 (0.2%) | 0 | 0 |
| Total | 13,464 (41%) | 17,106 (52%) | 66 (50%) | 239 (83%) |

Power Consumption is an essential value to be taken into account in embedded systems as long as it determines the autonomy and/or the operating cost of the system. Total Power Consumption of FPGAs is composed of a static component and a dynamic component. The first one is due to the transistor leakage current and the second one is based on the switching frequency of used FPGA resources. In modern FPGAs the static component is by far the most important because of the tiny size of the transistors in the current integration technologies. In the proposed design, the power has been analyzed with the *Xilinx's Power Analyzer* tool and a value of 3.720 W is calculated for the whole system for a 100 MHz frequency, being the static and dynamic components 3.071 W and 0.649 W respectively. As can be seen, it is a rather reasonable value for such an embedded system.

*Software parameters and execution times:* The programs to be executed in MB1 and MB2 occupy 700 K and 15.7 K respectively. The program for MB1 is clearly larger because it contains not only the training algorithms but also large data structures to handle large sets of training data. This code exceeds the size of the internal RAM so an external RAM is necessary, as we have already explained. However, to reduce the effect on speed that an external RAM program has, the code itself (39 K) has been stored in the internal RAM leaving data structures in the external RAM. Although a first off-line training could be performed previously in a standard computer, the data collection is still required whenever an on-line training has to be realized. Therefore, external RAM is mandatory in this design unless a much larger FPGA, endowed with large internal memory, is used.

The complete system (the 2 MicroBlazes, the PWM-Cores and the rest of the modules) operates at 100 MHz. With this clock frequency, each PWM-Core requires only 120 ns to execute an inference because, as we have described above, only 12 clock-cycles are necessary for the Core to complete the operation (i.e.: 12/100 MHz). As long as the 4 Cores can operate in parallel, more than 2 Million inferences per second can be achieved by the system which is really a very huge number. This is also possible because MB2 is dedicated almost exclusively to feeding the Cores with the incoming data. Note that a modern high-performance CPU would be able to perform an inference also in a very short time. For example, we made a comparison with an Intel Core 2 Duo E7660 CPU running at 3.06 GHz and we measured only 420 ns for an inference (about four times larger). However, unlike the CPU, numerous PWM-Cores can be implemented in the FPGA without increasing the computation time, since they all operate in parallel (4 cores in the example). Therefore, it would be necessary several high-end CPUs to obtain the same performance that the FPGA achieves. In addition, some other factors have to be taken into account such as the high power required for those high-performance CPUs with clock speeds above several GHz. Thus for example, the same above Intel Core (which is an expensive chip)

**Table 4**
Average time (ms) of a training iteration for different data set sizes.

| Points | 50 | 100 | 150 | 200 | 250 | 300 | 400 |
|---|---|---|---|---|---|---|---|
| Time | 430 | 776 | 1116 | 1457 | 1798 | 2139 | 2819 |

has a power consumption above 65 W even in idle state, while the proposed design requires only 3.72 W as it has been above explained.

On the contrary, MB1 takes much longer to complete a learning iteration (LSE + BP). Moreover the larger the training data size, the longer the time required. Table 4 shows the average times required to complete an iteration for different data set sizes (from 50 to 400 training points). As we can see, the times required for MB1 and MB2 are vastly different. This big time difference is one of the reasons for the use of 2 processors as was explained in Section 3. This means that in the case of an architecture with just only 1 microprocessor (for both training and inference), the inferences would have to be stopped for tens of seconds or even minutes whenever an on-line training was required (i.e. many iterations are required in a training stage. See Fig. 3). This can be a serious drawback in some Intelligent Environments where the system must be interacting permanently with the users or when the system is performing a critical task which cannot be stopped for a while. However, this problem is solved by the two-microprocessor architecture proposed in this work, and this represents one of the main advantages of our system.

Note also here, that a modern high-performance CPU can perform the training process much faster than a Microblaze at 100 MHz. But the learning process is here not as time-critical as the real-time response (i.e. MB2). Furthermore, the advantage of the proposed solution is that the two processes (i.e., training and real-time response) are integrated in a single device which implies a drastic reduction on power, size and cost. These high restrictions are required in Intelligent Environments as it is explained in Section 1.

## 6. Conclusions

In this work we have developed an FPGA-based embedded system for Intelligent Inhabited Environments. The Intelligent capabilities are addressed by means of an PWM-ANFIS system which is a simplified ANFIS-like Neuro-Fuzzy system specially suited for efficient implementations. On the other hand, the developed architecture is based on two soft-core microprocessors which perform the learning tasks and the on-line response respectively. In addition, this last microprocessor is attached with high performance hardware modules to accelerate the Neuro-Fuzzy system computations. To achieve this goal, the modules have been designed carefully taking into account the high parallelism that FPGAs exhibit.

To validate our proposal, different tests have been carried out with data from a real experiment: i.e., the "Intelligent Dormitory (iDorm)" which is a real ubiquitous computing test bed from Essex University [13]. First, the learning and adapting ability of the PWM-ANFIS has been proved by means of different training tests. All these tests show that this low-complexity version of the ANFIS retains quite good modelling ability even with high dimensionality systems. This property is essential for developing embedded systems with intelligent and powerful features but which are small enough to be attached in an Intelligent Inhabited Environment.

In addition, the system has been implemented using the XC5VSX50T FPGA of Xilinx' Virtex 5 family. Obtained values of resource utilization, power consumption and execution times confirm the validity of our proposal compared to microprocessor-based solutions. Moreover, the different execution times to perform the learning algorithms and the on-line response show the advantages of using two microprocessors in the proposed architecture. Furthermore, future works can analyze the utilization of as many microprocessors as PWM-ANFIS systems in order to minimize the time delays due to the multiple learning processes.

## Acknowledgements

This work was supported in part by the Spanish Ministry of Science and Innovation and European FEDER funds under Grant TEC2010-15388, and by the Basque Country Government under Grants IT419-10, S-PC10UN09 and S-PC11UN012.

## References

[1] European Commission IST Advisory Group, Scenarios for Ambient Intelligence in 2010, Final Report, February 2001.
[2] European Commission IST Advisory Group, Ambient Intelligence: From Vision to Reality, 2003.
[3] T. Basten, M. Geilen, H. de Groot (Eds.), Ambient Intelligence: Impact on Embedded System Design, Kluwer Academic Publishers, Boston, 2003 (Part I).
[4] M. Weiser, Some computer science issues in ubiquitous computing, Commun. ACM 36 (7) (1993) 74–84.
[5] B. Allen, I. Bierhoff, C. Bhler, E. Chandler, et al., Ambient Intelligent: Paving the Way John Gill Ed. COST Office, 2008.
[6] J.A. Kalomiros, J. Lygouras, Design and evaluation of hardware/software FPGA-based system for fast image processing, Microprocess. Microsyst. 32 (2008) 95–106.
[7] A. Astarloa, U. Bidarte, J. Lzaro, A. Aitzol, J. Arias, Multiprocessor SoPC-core for FAT volume computation, Microprocess. Microsyst. 29 (2005) 421–434.
[8] C.-T. Lin, C.S.G. Lee, Neural Fuzzy Systems; A Neuro-Fuzzy Synergism to Intelligent Systems, Prentice-Hall P T R, 1996.
[9] W. Pedrycz, F. Gomide, Fuzzy Systems Engineering: Toward Human-Centric Computing, John Wiley & Sons Inc., 2007.
[10] I. del Campo, J. Echanobe, G. Bosque, J.M. Tarela, Efficient hardware/software implementation of an adaptive neuro-fuzzy system, IEEE Trans. Fuzzy Syst. 16 (3) (2008) 761–778.
[11] J. Echanobe, I. del Campo, G. Bosque, J.M. Tarela, An adaptive neuro-fuzzy system for efficient implementations, Inform. Sci. 178 (2008) 2150–2162.
[12] I. del Campo, K. Basterretxea, J. Echanobe, G. Bosque, F. Doctor, A system-on-chip development of a neuro-fuzzy embedded agent for ambient-intelligent environments, IEEE Trans. Syst., Man, Cybernet., Part B: Cybernet. 24 (2) (2012) 501–512.
[13] F. Doctor, H. Hagras, V. Callahan, A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments, IEEE Trans. Syst., Man, Cybernet. – Part A 35 (1) (2005) 55–65.
[14] IEEE Pervasive Computing, IEEE Computer Society, ISSN: 1536–1268.
[15] Journal of Ambient Intelligence and Smart Environments, IOS-Press, ISSN: 1876–1364.
[16] W. Weber, J.M. Rabaey, E. Aarts, Ambient Intelligence, Springer, 2004.
[17] J.C. Augusto, D. Shapiro, Advance in Ambient Intelligence, IOS-Press, 2007.
[18] S. Poslad, Ubiquitous Computing, John Wiley & Sons Ltd., 2009.
[19] H. Nakashima, H. Aghajan, J.C. Augusto, Handbook of Ambient Intelligence and Smart Environments, John Wiley & Sons Ltd., 2009.
[20] P. Remagnino, G.L. Foresti, Ambient intelligence: a new multidisciplinary paradigm, IEEE Trans. Syst., Man, Cybernet. – Part A 35 (1) (2005) 55–65.
[21] A.V. Vasilakos, Special issue: ambient intelligence, Inform. Sci. 178 (3) (2008).
[22] C. Ramos, J.C. Augusto, D. Shapiro, Ambient intelligence-the next step for artificial intelligence, IEEE Intell. Syst. 23 (2) (2008).
[23] International Conferences on Intelligent Environments. <http://www.intenv.org/>.
[24] International Joint Conferences on Ambient Intelligence. <http://www.ami-conferences.org/>.
[25] A. Vainio, M. Valtonen, J. Vanhala, Proactive fuzzy control and adaptation methods for smart homes, IEEE Intell. Syst. 23 (2) (2008) 42–49.
[26] P. Rashidi, D.J. Cook, Keeping the resident in the loop: adapting the smart home to the user, IEEE Trans. Syst., Man, Cybernet. – Part A 39 (5) (2009) 949–959.
[27] F. Sadri, Ambient intelligence: a survey, ACM Comput. Surv. 43 (4) (2011).
[28] Diane J. Cook, Juan C. Augusto, Vikramaditya R. Jakkula, Ambient intelligence: technologies, applications, and opportunities, Pervasive Mobile Comput. 5 (2009) 277298.
[29] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, H. Duman, Creating an ambient-intelligence environment using embedded agents, IEEE Intell. Syst. 19 (6) (2004) 12–20.
[30] G.M.P. OHare, S. Keegan, M.J. OGrady, Realizing the ambient intelligence vision through the deployment of mobile, intentional agents, in: Proceedings of 2nd European Symposium on Ambient Intelligence (EUSAI 2004), Eindhoven University of Technology, Eindhoven, The Netherlands, November, 2004, Lecture Notes in Computer Science (LNCS), vol. 3295, Springer-Verlag, pp. 339–350.

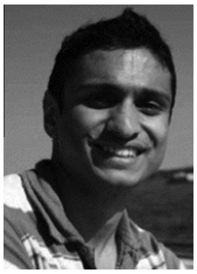
[31] N. Heo, P.K. Varshney, Energy-efficient deployment of intelligent mobile sensor networks, IEEE Trans. Syst., Man, Cybernet. – Part A 35 (1) (2005) 78–92.
[32] C. Papatheodorus, G. Antoniu, A. Bikakis, On the deployment of contextual reasoning in ambients intelligence environments, in: 2010 Sixth International Conference on Intelligent Environments (IE), Kuala-Lumpur, Malasya, July 2010, pp. 13–18.
[33] T. Nieberg, S. Dulman, P. Havinga, L. van Hoesel, J. Wu, Collaborative algorithms for communication, in: T. Basten, M. Geilen, H. de Groot (Eds.), Wireless Sensor Networks, in Ambient Intelligence: Impact on Embedded System Design, Kluwer Academic Publishers, Boston, 2003, pp. 271–294.
[34] R. Min, A. Chandrakasan, Energy-efficient communication for high density networks, in: T. Basten, M. Geilen, H. de Groot (Eds.), Ambient Intelligence: Impact on Embedded System Design, Kluwer Academic Publishers, Boston, 2003, pp. 295–314.
[35] Computational Intelligence (Special Issue), Proceedings of the IEEE, September 1999.
[36] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 1999.
[37] D.B. Fogel, Evolutionary Computation. Toward a New Philosophy of Machine Intelligence, IEEE Press, Piscataway, NJ, 1995.
[38] L. Aguilar, P. Melin, O. Castillo, Intelligent control of a stepping motor drive using a hybrid neuro-fuzzy ANFIS approach, Appl. Soft Comput. 3 (3) (2003) 209–219.
[39] O. Cordon, F. Herrera, F. Hoffmann, L. Magdalena, Genetic fuzzy systems. Evolutionary tuning and learning of fuzzy knowledge bases, in: Advances in Fuzzy Systems: Applications and Theory. World Scientific, 2001.
[40] Y. Maldonado, O. Castillo, P. Melin, Particle swarm optimization of interval type-2 fuzzy systems for FPGA applications, Appl. Soft Comput. 13 (1) (2013) 496–508.
[41] F. Taeed, Z. Salam, S.M. Ayob, FPGA implementation of a single-input fuzzy logic controller for boost converter with the absence of an external analog-to-digital converter, IEEE Trans. Ind. Electron. 59 (2) (2012) 1208–1217.
[42] R. Sepulveda, O. Montiel, O. Castillo, P. Melin, Embedding a high speed interval type-2 fuzzy controller for a real plant into an FPGA, Appl. Soft Comput. 12 (3) (2012) 988–998.
[43] O. Montiel, J. Camacho, R. Seplveda, O. Castillo, Embedding a fuzzy locomotion pose controller for a wheeled mobile robot into an FPGA, Soft Comput. Intell. Contr. Mobile Robot. (2011) 465–481.
[44] T. Matsubara, H. Torikai, Asynchronous cellular automaton-based neuron: theoretical analysis and on-FPGA learning, IEEE Trans. Neural Netw. Learn. Syst. 24 (5) (2013) 736–748.
[45] Markos Papadonikolakis, Christos-Savvas S. Bouganis, Novel cascade FPGA accelerator for support vector machines classification, IEEE Trans. Neural Netw. Learn. Syst. 23 (7) (2012) 1040–1052.
[46] Tesresa Orlowska-Kowalska, Marcin Kaminski, FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system, IEEE Trans. Ind. Inform. 7 (3) (2011) 436–445.
[47] Cheng-Hao Huang, Wen-June Wang, Chih-Hui Chiu, Design and implementation of fuzzy control on a two-wheel inverted pendulum, IEEE Trans. Ind. Electron. 58 (7) (2011) 2988–3001.
[48] Denis Vinicius Coury, Alexandre Cludio Botazzo Delbem, Janison Rodrigues De Carvalho, Mrio Oleskovicz, Eduardo V. Simes, Daniel Ignacio Barbosa, Tiago V da Silva, Frequency estimation using a genetic algorithm with regularization implemented in FPGAs, IEEE Trans. Smart Grid 3 (3) (2012) 1353–1361.
[49] Shing-Tai Pan, Xu-Yu Li, An FPGA-based embedded robust speech recognition system designed by combining empirical mode decomposition and a genetic algorithm, IEEE Trans. Instrum. Meas. 61 (9) (2012) 2560–2572.
[50] Ching-Chili Tsai, Hsu-Chih Huang, Shui-Chun Lin, FPGA-based parallel DNA algorithm for optimal configurations of an omnidirectional mobile service robot performing fire extinguishment, IEEE Trans. Ind. Electron. 58 (3) (2011) 1016–1026.
[51] Pradeep R. Fernando, Srinivas Katkoori, Didier Keymeulen, Ricardo Salem Zebulum, Adrian Stoica, Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine, IEEE Trans. Evol. Comput. 14 (1) (2010) 133–149.
[52] R.A. Ramadan, H. Hagras, M. Nawito, A. El Faham, B. Eldesouky, The intelligent classroom: towards an educational ambient intelligence testbed, in: Proc. 2010 Sixth International Conference on Intelligent Environments, Kuala Lumpur, 2010, pp. 344–349.
[53] A.I. Dounis, C. Caraiscos, Advanced control systems engineering for energy and comfort management in a building environmental review, Renew. Sustain. Energy Rev. 13 (2009) 12461261.
[54] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A hierarchical fuzzygenetic multi-agent architecture for intelligent buildings online learning, adaptation and control, Inform. Sci. 150 (12) (2003) 3357.
[55] J.-S. Jang, ANFIS: adaptive-network-based fuzzy inference system, IEEE Trans. Syst., Man Cybernet. 23 (1993) 665–685.
[56] Virtex 5 Family Overview, Xilinx Inc., San Jose, CA, 2009. <http://www.xilinx.com/support/documentation/datasheets/ds100.pdf>.
[57] ISE Design Suite, Xilinx Inc., San Jose, CA, 2009. <http://www.xilinx.com/tools/designtools.htm>.
[58] ModelSim, Advanced Simulation and Design, Mentor Graphics Corporation, Wilsonville, OR. <http://model.com/>.

**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. He has been an Associate Professor in the Department of Electricity and Electronics, UPV/EHU, from 1999 to 2009. Since 2009 he is a Senior Lecturer in that department. His research interests focus on (1) digital electronics: embedded systems, reconfigurable FPGAs, DSPs, SoPC; (2) computational intelligence: artificial neural networks, fuzzy systems, and neuro-fuzzy systems; (3) ubiquitous computing: ambient intelligence, intelligent environments. Dr. Echanobe has published many papers in international journals and conferences in most of those areas.

**Inés del Campo** (A'96–M'11) was born in Buenos Aires, Argentina, in 1961. She received the Licenciado degree in physics with specialization in electronics and automatics in 1987 and the Ph.D. degree in physics, in 1993, both from the University of the Basque Country (UPV/EHU), Bilbao, Spain.
Currently she is a Senior Lecturer in the Electricity and Electronics Department of the Faculty of Sciences and Technology of the UPV/EHU. She has published articles in international journals and conferences in the areas of electronics, computational intelligence, intelligent control, ambient intelligence, and pattern recognition, among others. Her research interests mainly concern system-on-chip (SOC) design, hardware/software codesign, reconfigurable hardware, pervasive computing, artificial neural networks (ANNs), fuzzy systems, and genetic algorithms. She is also interested in the internet of things and its application in the context of ubiquitous computing and ambient intelligence.

**Koldo Basterretxea** (M'02) was born in Bilbao, Basque Country, Spain, in 1970. He received the Licenciado degree in physics with specialization (M.Sc.) in electronics and control in 1994 and the Ph.D. degree in physics in 2002, both from the Universidad del País Vasco/Euskal Herriko Unbertsitatea (UPV/EHU), Bilbao, Basque Country, Spain.
He was a Lecturer at the Electronics and Telecommunications Department at the Escuela Universitaria de Ingeniería Técnica Industrial (EUITI) of Eibar (UPV/EHU), from 1995 to 1998, and at the EUITI of Bilbao (UPV/EHU) since 1998, where he currently is a Senior Lecturer. His research interests include digital design of adaptive systems on FPGAs, hardware design for high-speed real-time controllers, neural/fuzzy hardware, hardware/software co-design, and applied soft computing.

**Maria Victoria Martínez** was born in Bilbao, Spain in 1964. She received the Licenciado degree in physics with specialization in electronics and automatics in 1987 and the Ph.D. degree in physics, in 2002, both from the University of the Basque Country (UPV/EHU), Bilbao, Spain.
She was a Predoctoral Researcher (granted by the Basque Government) from 1987 to 1998. She has been since 1995 an Associate Professor in the Department of Electricity and Electronics, UPV/EHU. She has published articles in international journals and conferences in the areas of electronics, computational intelligence, device modeling, ambient intelligence, among others. Her research interests mainly concern the synthesis and electronic implementation of piecewise linear (PWL) systems with emphasis on efficient realizations for highly complex systems.



**Faiyaz Doctor** received the B.Sc. degree in computer science and artificial intelligence from the School of Computer Science, University of Birmingham, Birmingham, UK, in 1998 and the M.Sc. degree in computer science and artificial intelligent agents in 2002 and PhD in computer science in 2006 from the School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK.
After completing his PhD he has worked in both industry and within the academic community to develop novel artificial intelligence solutions for addressing real world problems related to smart environments, energy optimization, predictive analytics and decision support. His work has resulted in high profile innovation awards and an international patient for improved approaches for data analysis and decision-making using hybrid neuro-fuzzy and type-2 fuzzy systems. He is currently a lecturer in Computing with the faculty of Engineering and Computing, Coventry University, UK. His research interests include computational intelligence, fuzzy logic, applications of type-2 fuzzy logic, hybrid systems using fuzzy logic with genetic algorithms and neural networks, ambient intelligence, pervasive computing and intelligent buildings. Other areas of interest also include embedded agents, intelligent machines and applications of computational intelligence in business, commerce and healthcare. Dr. Doctor has published a number of papers in international journals and conferences in the field of pervasive computing and computational intelligence.