Hardware/software codesign of configurable fuzzy control systems

A. Cabrera [a], S. Sánchez-Solano [b], P. Brox [b], A. Barriga [b,*], R. Senhadji [b]

[a] *Dpto. Automática y Computación, Facultad de Ingeniería Eléctrica, ISPJAE, Calle 127 s/n, Marianao, Ciudad de la Habana, Cuba*
[b] *Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica, CSIC, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain*

Abstract

Fuzzy inference techniques are an attractive and well-established approach for solving control problems. This is mainly due to their inherent ability to obtain robust, low-cost controllers from the intuitive (and usually ambiguous or incomplete) linguistic rules used by human operators when describing the control process. This paper focuses on the hardware/software codesign of configurable fuzzy control systems. Two prototype systems implemented on general-purpose development boards are presented. In both of them, hardware components are based on specific and configurable fuzzy inference architecture whereas software tasks are supported by a microcontroller. The first prototype uses an off-the-shelf microcontroller and a low-complexity Xilinx XC4005XL field programmable gate array (FPGA). The second one is implemented as a system on programmable chip (SoPC), integrating the microcontroller together with the fuzzy hardware architecture and its interface circuits into a Xilinx Spartan2E200 FPGA.

Keywords: Fuzzy controllers; Codesign techniques; SoPC; FPGA

1. Introduction

Fuzzy logic-based inference techniques provide a practical mechanism for describing the behavior of a complex system by means of linguistic terms similar to those employed in natural language. The capability of fuzzy controllers to capture the knowledge of a human expert and translate it into a robust control strategy without a mathematical model of the system under control has motivated a great increase in the number of control applications using these techniques in the last years [1–4].

Although they can be the base for approximate reasoning mechanisms, fuzzy inference techniques provide systematic and deterministic algorithms that can be easily programmed in a general-purpose processor or implemented as an electronic circuit. Thus, there are basically two alternatives for fuzzy controller implementation: one of them is based on software and the other on hardware. The software solution offers flexibility as one of its main features: the designer can choose any type of fuzzy sets, operators, and rule bases. Many fuzzy controllers have been implemented in software on general-purpose computers by describing the fuzzy algorithm with a high-level programming language. In applications where there are cost, weight, size, or power constraints, as occurs with consumer electronics, standard microcontrollers have been usually employed [2,4]. Both types of software implementations (microcomputer- and

* Corresponding author. Tel.: +34-9550-56-666; fax: +34-9550-566-86.
E-mail addresses: alex@electrica.cujae.edu.cu (A. Cabrera), santiago@imse.cnm.es (S. Sánchez-Solano), barriga@imse.cnm.es (A. Barriga).

microcontroller-based) have the same speed limitation due to the inherent sequential program execution, so that they are not adequate for many real-time control applications and a hardware approach must be used. Full hardware implementations achieve a very high inference speed but are characterized by their lack of flexibility because some limitations must be adopted in order to obtain a cost-effective dedicated hardware. Several microelectronics implementations of fuzzy controllers have been proposed in the last few years [5]. Both, digital and analog architectures have been developed, with the digital ones being the most widespread because their design is facilitated by the use of well-established design methodologies and the advances in computer aided design (CAD) tools.

Halfway between full software and full hardware implementations, hardware/software (HW/SW) codesign techniques try to obtain an appropriate trade-off between the advantages and drawbacks of both approaches [6]. In particular, the application domains we are dealing with, such as embedded, real-time, and reactive systems, are the applications areas for which HW/SW codesign techniques are most effective. The hardware–software partitioning of the tasks to be carried out by the fuzzy controller allows for both a flexible and high-speed system. Higher time-consuming tasks should be implemented in hardware while those related to system configuration are better executed by software. As a result, an efficient low-cost fuzzy controller can be developed by means of codesign methods.

The rapid transition in silicon technologies has not only increased the number of devices in application-specific integrated circuits (ASICs) but has also allowed programmable logic devices like FPGAs to increase in density and reduce costs, so that hardware platforms based on FPGAs can be used for rapid prototyping approaches as well as for final solutions which greatly shorten the time-to-market of new products. The current availability of low-cost large-capacity FPGAs, an increasing number of circuit components described as intellectual property (IP) modules, and powerful CAD tools make it possible to develop a whole system on a programmable chip (SoPC).

The above considerations have led us to develop fuzzy control systems by means of a codesign method and based on FPGA platforms. This paper presents two of these developments in which specific hardware components for the fuzzy controller are designed according to a configurable architecture for fuzzy inference systems whereas software tasks are supported by microcontrollers. One of the systems uses a small-size Xilinx XC4005XL FPGA and an off-the-shelf microcontroller. The second one is a fuzzy controller implementation as a SoPC which integrates the microcontroller and the fuzzy hardware architecture into a Xilinx Spartan2E200 FPGA.

The paper is organized as follows. After describing briefly the structure of a fuzzy control system and the tasks it has to carry out, Section 2 summarizes the codesign alternatives to implement these tasks in a hardware/software platform. The alternative we have followed is described in detail in Section 3, including the specific architecture selected for the hardware component. The designs of two FPGA-based fuzzy control systems are explained in Sections 4 and 5. Finally, some conclusions are given in Section 6.

2. Codesign alternatives for fuzzy control systems

The generic structure of a closed-loop fuzzy control system, including the controlled process, is shown in Fig. 1. A fuzzy controller employs the same input and output variables as its conventional counterpart. The difference between both approaches is that in the fuzzy case the control heuristics is defined by a rule set which employs linguistic variables represented by fuzzy sets, instead of being obtained as a linear combination of the inputs. A set of sensors are in charge of acquiring the variables defining the process status and translating them into electrical signals. Analog-to-digital (A/D) conversion (if necessary), range adjustment, and pre-processing algorithms required to obtain adequate input signals for the inference engine are carried out by the signal conditioning input stage. In order to calculate the controller output, the fuzzy inference module (FIM) must fuzzify the inputs, calculate the new control action according to the strategy defined by the set of inference rules, and compute the non-fuzzy representative values of the control action through a defuzzifier. Finally, the signal conditioning output stage post-processes these values and adapts them to the actuators, usually through digital-to-analog (D/A) converters. In addition to these basic tasks, the control

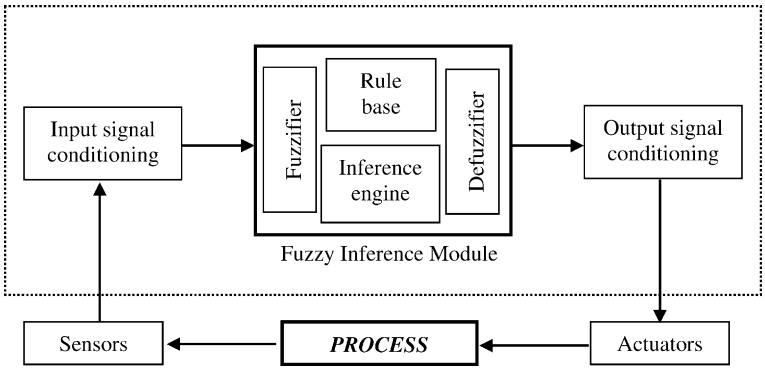


Fig. 1. Block diagram of a fuzzy logic-based control system.

system should also include elements to initialize its operation, to define the control targets, and to monitor the achievement of the targets, as well as timing elements to ensure the correct behavior of the system.

Traditionally, all of the above-mentioned tasks, including the inference mechanism, have been performed by software on a programmable processor. Many applications for industrial or consumer products resort to this type of implementation [2–4]. However, when the application demands fuzzy control systems with a high-inference speed and low size and power, a more efficient solution using dedicated hardware must be adopted, but this solution limits greatly the controller flexibility. Since neither the standard full software approaches nor the specific full hardware ones are able to meet all of these requirements, several codesign solutions have been reported. The codesign process of any system has to analyze all the tasks that the system must carry out, evaluating the impact of the possible implementation options over the factors defining the cost and global system functionality [6]. The main parameters to consider in this evaluation are the task execution speed and the area required by its hardware implementation. From the previous results, a partitioning process is carried out to decide which tasks should be executed by software and which ones should be implemented on hardware.

There are different approaches for applying codesign techniques in the development of control systems based on fuzzy logic. They can be distinguished by the form and level in which the HW/SW partitioning is made. One of them analyzes the influence of the instruction set of a processor over the implementation of a fuzzy inference system so as to redesign it in such way that it supports the operations which best contribute to increasing the inference speed. Taking into account that many of the operations required by the fuzzy inference to a great extend limit the operational speed of the fuzzy controller, the idea is to speed up the execution of operations such as the maximum, minimum, or defuzzification [7]. Thus, the processor microprogramming is modified in some cases in order to add functionality to the instruction set [8,9], while in other cases, new circuitry is included in the processor architecture [10–12]. In all of these cases, the partitioning process is carried out at the instruction level of the processor.

Another possibility is to implement the fuzzy inference module, totally or partially, by means of dedicated hardware with a specific architecture. This ‘a priori’ partition is based on the fact that, among all the tasks that a fuzzy logic-based control system must carry out, those related to the fuzzy inference calculation are the most timing consuming [7]. Therefore, their implementation by means of specific hardware contributes significantly to increase the controller speed. With this approach, the hardware-implemented FIM acts as a coprocessor of a general-purpose microcontroller which executes the rest of the tasks implemented by software (initialization, input and output processing, etc.). The hardware-implemented FIM obtains the inference conclusions very fast and communicates them back to the microcontroller which closes the control loop (Fig. 2).

The previous codesign variant is the one we have followed in the controllers described in this paper: the

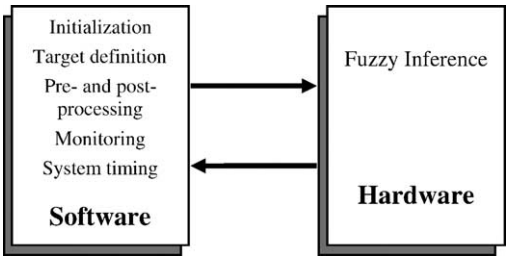


Fig. 2. Codesign-based fuzzy controller macrostructure.

complete fuzzy inference module is implemented in programmable hardware (FPGA development boards) following a specific and efficient architecture whereas the remaining control tasks are programmed in software and executed by a standard microcontroller. The following section describes in detail our codesign approach.

3. A codesign methodology for fuzzy control systems

The design cycle of a microelectronic system can be greatly accelerated by the use of a methodology which defines the different design stages and their interrelations. Following a conventional codesign methodology, the design starts with a system specification stage which considers the requirements of the control application. User-friendly specifications and high-level formal languages are usually employed at this stage and the whole control system is simulated to evaluate and verify the system specification roughly. The hardware/software partitioning is performed at the next design stage so as to decide which components of the system will be realized in hardware and which will be implemented in software. The third stage aims at synthesizing the hardware components, software modules, and hardware/software interfaces. Finally, all the modules are integrated and co-simulated to verify the system behavior in detail.

Another fact which greatly accelerates the system design process is the use of CAD tools which make the carrying out of the tasks at the different stages easier. When designing a fuzzy control system, the central component which has to be well-specified is the fuzzy inference module. Although the capability of fuzzy logic is exploited to translate expert knowledge expressed linguistically into an inference algorithm expressed mathematically, this translation is not easy because there are many ways to do it. For example, different membership functions can be chosen to represent the fuzzy sets associated with the linguistic variables, several defuzzification methods can be selected to obtain a non-fuzzy conclusion, etc. Hence, a very important consideration is the availability of CAD tools for the development of the fuzzy inference module. Several CAD environments have been reported in the literature for this purpose. We have used the *Xfuzzy* environment [5,13], because it includes tools not only for specifying a fuzzy system but also for relating it to the other control modules, for generating software or hardware descriptions, and for evaluating the control system performance, so as to carry out a complete codesign methodology [14,15]. The flow of this codesign methodology and the tools of *Xfuzzy* employed (*xfc*, *xfsim*, *xflab*, and *xfvhdl*) are shown in Fig. 3. The methodology steps are explained in the following.

3.1. System specification and verification

Xfuzzy eases the description of the fuzzy inference module with the XFL language, a formal specification language for fuzzy systems which is shared by all the tools of the *Xfuzzy* environment [16]. The XFL language allows the user to define the universes of discourse of the fuzzy controller variables, their associated membership functions, and the rule bases employed, as well as to choose the set of fuzzy operators which implement the antecedent connection, the implication function, the mechanism for rule aggregation, and the defuzzification method. The graphical user interfaces (GUIs) provided by *Xfuzzy* make this definition more comfortable. Fig. 4 shows the interfaces employed to define the membership functions and singleton values associated to the input and output variables of a fuzzy controller.

In order to verify the behavior of the whole control system, the “XFL to C compiler” (*xfc*) tool included in the *Xfuzzy* environment is employed to obtain a C description of the fuzzy inference module from its XFL specification. The code generated by *xfc*, together with the programming code associated to

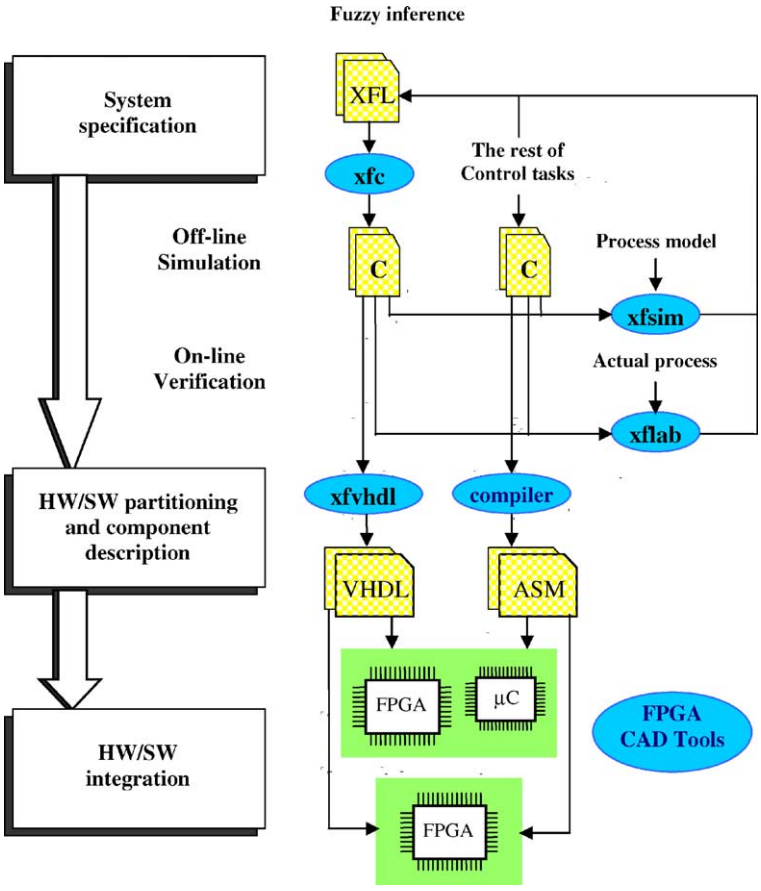


Fig. 3. Codesign flow and CAD tools for FPGA-based fuzzy controllers.

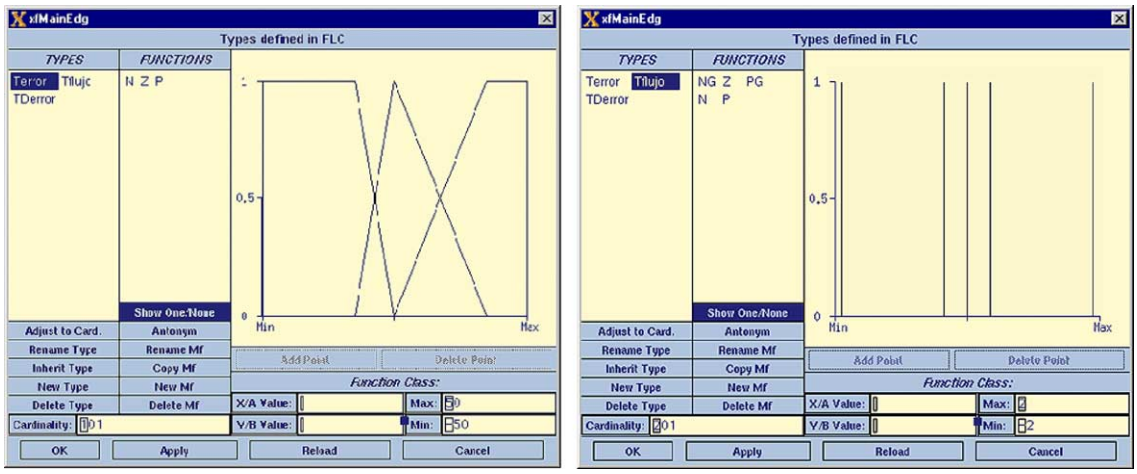


Fig. 4. GUIs provided by *Xfuzzy* to define membership functions of linguistic variables.

the rest of the control tasks, can be combined with a C model of the process under control to simulate the behavior of the closed-loop control system. This simulation, which is known as off-line simulation, is carried out within the *Xfuzzy* environment using the *xfsim* tool. This simulation tool is very useful to obtain a preliminary adjustment of the control system parameters. However, since the model used to represent the process is usually a first-order approximation, the results of this simulation could not be reliable enough for control systems of a certain complexity. In these cases, the actual process may be analyzed to obtain a better adjustment of the control system parameters. One solution we have explored is to include the actual process into the control loop where the whole control system, including the FIM, is implemented by software over a computer running a C program. This is known as on-line verification. It is important to notice that this verification is possible whenever the response time of the computer meets the process dynamic requirements.

The tool *xflab* provides the needed mechanisms for communicating the computer which runs the C code of the fuzzy control system with the process under control via a data acquisition board connected to the internal bus of the computer [17]. *Xflab* contains drivers to read and write data from and in the board, so that the model of the process used in the off-line simulation can be replaced by a function which allows us to monitor the true process, thus obtaining the required information to execute the fuzzy control, as well as acting over the process providing the control action via the output channels. *Xflab* eases the configuration of the data acquisition board, the codification of the required pre- and post-processing procedures, and the integration of these tasks with both the software implementation of the controller and the routines to access the actual process. One of the most relevant advantages offered by this tool is the capability of monitoring the behavior of the whole system in real time, allowing the evaluation of a great variety of operation conditions as well as the influence of the different control parameters on the system efficiency.

Once the control system has been adjusted conveniently, a software implementation of the control system which is quite operative is available to proceed to the next methodology step.

3.2. HW/SW partitioning and component description

In the hardware/software partitioning stage the different tasks identified in the specification phase must be assigned to hardware or software resources and each of these components must be conveniently defined. The partitioning of the different tasks to be performed by the control system is done according to the considerations mentioned in Section 2: the FIM will be implemented in hardware with a specific circuit architecture, while the routines for processing the input and output variables of the fuzzy controller, together with all the other control tasks, will be implemented in software on a microcontroller. The use of development tools and program debuggers (assemblers, compilers, simulators, etc.) eases the programming of the tasks to be performed by the microcontroller. The availability of C compilers for the chosen microcontroller is a factor to take into account since it reduces greatly the development time. The memory capacity of the microcontroller must also be considered.

The design of the hardware components for the fuzzy system can be accelerated with the employment of specific tools. *Xfuzzy* is distinguished from other fuzzy environments by its development tools for hardware support. The *xfvhdl* tool included in *Xfuzzy* allows us to obtain a VHDL description of a fuzzy inference module starting from its XFL specification [18]. *Xfvhdl* uses a cell library containing the parameterized VHDL description for a set of basic building blocks required to implement an efficient specific architecture [19].

An efficient architecture is required to meet restrictive requirements in terms of speed, size, and/or power. Some key points to achieve low-cost and high-speed digital fuzzy inference modules are: (a) to evaluate only the contribution of the active rules; (b) to restrict the shapes of the fuzzy set membership functions; and (c) to use simplified defuzzification methods [19]. The overall impacts of these considerations are the following.

(a) Active rule processing means that only those rules which contribute to the final solution are processed instead of the whole rule memory. Consider an "I" input fuzzy controller, with "L" fuzzy sets with a "D" overlapping degree. The total number "R_t" of fuzzy rules is $R_t = L^I$. However, the number

"R_a" of active rules is only $R_a = D^I$. For instance, if the fuzzy controller has two inputs and seven fuzzy sets per input with an overlapping degree of 2, an active rule-driven architecture must only process 4 instead of 49 rules.

(b) Constraining the shapes of the input fuzzy sets by limiting its overlapping degree makes it possible to reduce the number of active rules as mentioned above. In addition, if the input fuzzy sets are also constrained to normalized piecewise-linear functions, the membership function degree can be obtained by arithmetic methods and also the defuzzification process can be simplified if product is used as the rule-antecedent connective operator.

(c) Traditional defuzzification methods, usually employed in software implementations, must sweep the whole universe of discourse in order to obtain the output crisp value. Hardware realization of such methods leads to the intensive use of parallel architectures or sequential techniques but both solutions are inefficient in terms of circuit area consumption or inference speed. Thus, the use of simplified defuzzification methods in which the conclusion obtained from a fuzzy rule is represented by the rule consequent "c_i" and its weight "w_i", must be adopted. Using simplified defuzzification methods makes it possible to obtain the output crisp value "y" as the average of the different conclusions weighted by the activation degrees "h_i" of the corresponding rules, as expression (1).

$$y = \frac{\sum_i h_i c_i w_i}{\sum_i h_i w_i} \tag{1}$$

The meaning of w_i in (1) leads to different defuzzification methods. The fuzzy mean method (FM) is the simplest one and it is obtained if $w_i = 1$. This method does not consider the area and support of the output fuzzy sets. Others simplified defuzzification methods such as the weighted fuzzy mean (WFM) and the quality method (QM) can be achieved giving different meanings to the w_i parameter [5,20].

A configurable architecture suitable for an efficient hardware realization of fuzzy inference modules based on the above concepts is depicted in Fig. 5. The membership function circuits (MFCs) determine the active antecedents for each input value, and provide as many pairs (label, activation level) as overlapping degree has been fixed for the system. The inference process is carried out by processing sequentially the active rules by means of an active rule selection circuit composed of a counter-controlled multiplexer array. In each clock cycle, the membership degrees "μ_i" of rule antecedents are combined within the connective-antecedent operator circuit to calculate the activation level of the rule "h_i", while the antecedent labels "L_i" address the position in the rule memory containing the parameters (c_i, w_i) which define the corresponding consequent. Finally, a configurable defuzzifier block is used to obtain the output crisp value [19].

System configurability is achieved by choosing different circuit level solutions to implement the building blocks of Fig. 5. Membership function circuits can be built resorting to either memory-based or arithmetic approaches. Memory-based MFCs store the labels and the membership degrees corresponding to each point of the input universe of discourse. Fig. 6 represents a memory-based MFC considering an overlapping degree of two. The total number of memory locations depends on the input resolution bits and each location stores the pair values of label-activation degree. The fuzzification process is carried out by a simple access to this memory using the input binary value as its address. Note that if labels "L_a" and "L_b" are binary coded, only one of them "L_a" must be included into each memory location because the other one can be easily obtained by adding "1". Memory-based MFCs allow us to define unrestricted fuzzy set shapes but each input needs a memory capacity, AM_i, expressed by (2).

$$AM_i = 2^B \times [2N + \lceil \log_2 L \rceil] \text{bits} \tag{2}$$

where B is the input resolution bits, N the bit number for the membership degree, and L is the number of fuzzy sets. Note that memory-based MFCs may consume a high circuit area unless the target device has dedicated resources for memory implementation.

When a reduced area is a primary target and there are no dedicated memory resources, an arithmetic block implementing normalized triangular functions may provide better results for MFC. Fig. 7 shows a schematic implementation for an arithmetic-based MFC. The intersecting points "a_i" and slopes "m_i" of the linear portions of each input are stored in a common parameter memory. Each input has an arithmetic

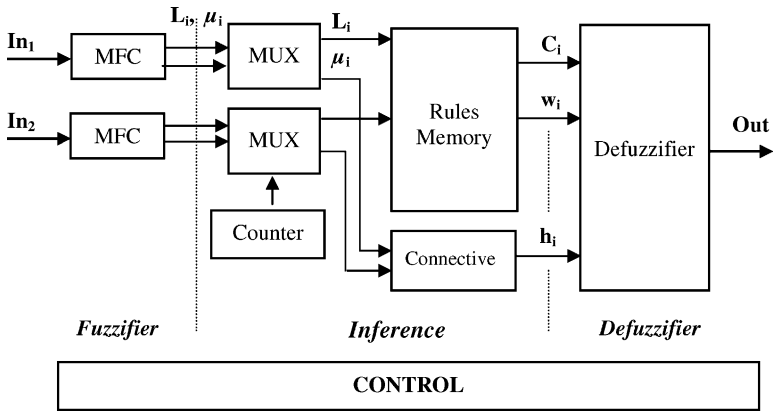


Fig. 5. Block diagram for the fuzzy inference module architecture.

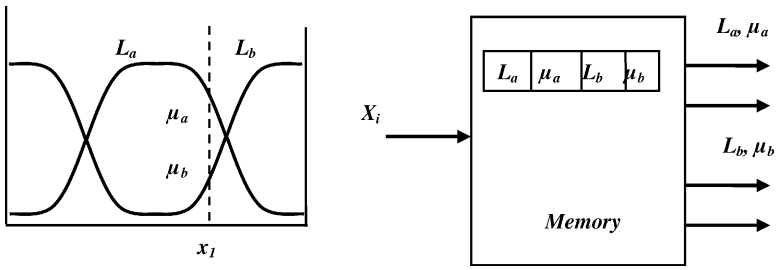


Fig. 6. Memory-based MFC illustration.

circuit which resolves a straight line equation in order to calculate the corresponding membership function degree. A counter accesses the parameter memory sequentially and the arithmetic circuit calculates and stores the membership function degree (and also the counter output) as long as the input value is greater than the intersection value. The last stored values correspond to the desired membership degree and its corresponding label. Due to the use of normalized functions, the second membership degree is easily obtained by a 1's complement circuit. The common parameter memory which is shared by all inputs has a capacity, PM, expressed by (3).

$$\mathrm{PM} = L[I(B+S)]\text{bits} \tag{3}$$

where "I" is the number of inputs, and "S" is the slope codification bits. Note that this memory has few locations but they are very wide. For instance, if 8 bits are used to codify all the parameters of Fig. 7 ($L = 4$, $I = 2$), a 4×32 bit memory is required. Thus, its

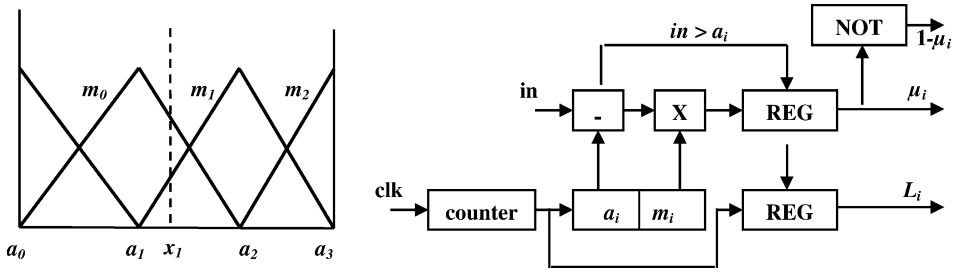


Fig. 7. Arithmetic-based MFC illustration.

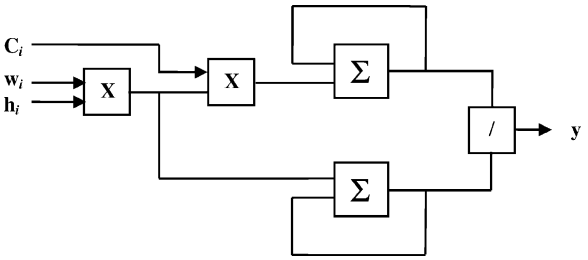


Fig. 8. Block diagram for a configurable defuzzifier module.

implementation on dedicated memory resources is not recommended unless these resources would remain unused.

There are also different options for implementing the defuzzifier block according to the simplified method chosen. A circuit composed of two multipliers, a divider, and two adders as shown in Fig. 8 may implement all the defuzzification methods described by (3). As a particular case, for fuzzy mean implementation ($w_i = 1$) only one multiplier is needed. Also, if normalized input membership functions are used and the product is selected as the antecedent connective operator, the denominator in expression (1) is always 1, and the divider circuit can be removed from the block diagram in Fig. 8.

Using *xfvhdl*, the VHDL code for the specific architecture described above is generated from the XFL specification of the fuzzy module. The architectural options (memory- or arithmetic-MFCs and defuzzification method) and the resolution bits are defined by the user when the tool is executed.

Once the hardware and software modules have been generated, the next methodology step is to integrate them.

3.3. HW/SW development and integration

The VHDL description of the fuzzy module is the starting point to implement it on an FPGA. In this step the FIM should be simulated to verify its performance again. A VHDL simulator such as *ModelSim* of Mentor Graphics can be used for this purpose. Next, we proceed to its logic synthesis and its implementation on the FPGA using specific CAD tools such as those provided by Synopsys (*FPGA Compiler 2* and *FPGA Express*) and Xilinx. To accelerate this design stage, the tool *xfvhdl* produces as output not only the VHDL files describing the fuzzy module but also a testbench file which eases the fuzzy module simulation and script files containing command procedures for batch executing the synthesis and implementation processes on Xilinx FPGAs. These files, together with those describing the interface of the fuzzy module with the microcontroller, are combined into a standard FPGA design flow, thus obtaining the hardware component implementation of the fuzzy control system.

The whole fuzzy control system can be integrated basically by following two approaches: (a) the hardware component implemented on an FPGA is connected to an off-the-shelf microcontroller which implements the software component; or (b) the hardware and software components are both implemented on an FPGA as a system on programmable chip (SoPC). The first approach allows rapid system integration thanks to the availability of FPGA development boards which include a general-purpose microcontroller. The codesign implementation of the fuzzy control system finishes by downloading the executable microcontroller program and the FPGA bit stream configuration file to the development board. A fuzzy controller based on this approach is described in Section 4.

The second approach allows for a more customized design, and, hence, better performance in terms of area, power, and speed can be obtained. This approach is currently facilitated by several facts. (a) the availability of large and powerful FPGA devices with millions of gates-equivalent inside. Some FPGA devices include not only general logic but also dedicated logic as memory structures, hardware multipliers, and even microprocessors. Using these components, a very complex system can be included into a programmable device. (b) The existence of many systems components available as intellectual property modules (such as interfaces and cores for processors or microcontrollers), which can be easily included in a design. (c) The application of design techniques for reusability (based in the intensive use of IP modules) which reduce the increased gap between integration and design capacities. Since the implementation details of different parts of a system can be reduced or avoided by using IP modules, designers can work more on the behavioral level of a system than on the structural one. (d) The existence of powerful CAD tools which ease the integration of component descriptions and IP modules with optimizations for a target programmable

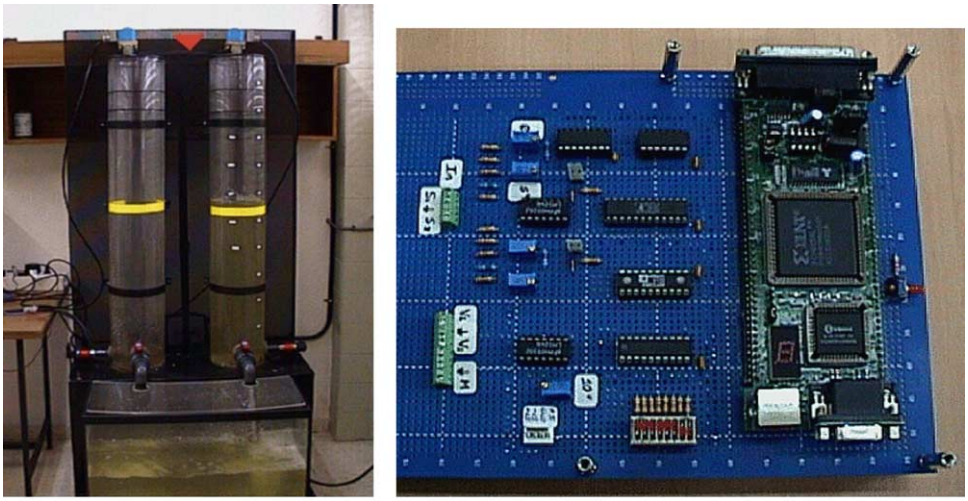
device. Section 5 describes how these facts have been applied in the codesign of a fuzzy controller as a SoPC.

4. A fuzzy controller using an off-the-shelf microcontroller

As mentioned above, one of the approaches for integrating a fuzzy control system is to connect the hardware component implemented on an FPGA to the software component implemented on an off-the shelf microcontroller. Among the development boards which include a general-purpose microcontroller with an FPGA, we have used an XS40-005XL board from XESS Corporation [21]. This board has an Intel 8031 microcontroller, a low-complexity Xilinx XC-4005 FPGA (196 configurable logic blocks, CLB), 32 Kbytes of SRAM, a programmable clock circuit, a seven-segment LED display, and connectors for communicating the board with a PC via a parallel port and other interfaces. The board is provided with software facilities to program the clock and to download the FPGA configuration and the microcontroller program among others.

To illustrate the above described codesign methodology with this approach, let us consider a typical control problem of designing a level controller for a dosage system by means of a fuzzy controller. The dosage system considered is shown in Fig. 9a. It is composed of two independent cylindrical tanks both with an electronic valve at the top which control the liquid injection, with a pressure sensor at the bottom to provide a measure of the liquid level, and with a manually-controlled valve also at the bottom which determines how fast the liquid is discharged into a shared container. A voltage-controlled water pump moves the liquid up to the tanks from the shared container. The pressure sensors provide a current output signal between 4 and 20 mA, while the electronic valves and the water pump are controlled by voltage input signals between 0 and 10 V. Additional conditioning circuitry (in particular A/D and D/A converters) have been employed to adapt the signals provided by the sensors to the development board as well as to adapt the output signals of the board to those employed by the actuators (Fig. 9b).

The control strategy applied is the fuzzy version of a PI controller with an incremental output. This means that the fuzzy controller requires two input signals (the error and its variation) for each tank, and provides an output signal which represents the change of its corresponding valve aperture. The water pump is governed by the bounded sum of both controller outputs. Due to the symmetry of the two tanks, their fuzzy inference modules are identical, so that we can implement



(a) (b)

Fig. 9. Experimental set up: (a) dosage system; (b) control system implementation.

in hardware only one of them if the inference processes are executed sequentially. This solution permits a great area reduction without increasing the response time significantly.

Starting from the XFL specification of the fuzzy inference module, off-line simulation and on-line verification stages were carried out with *xfsim* and *xflab*, respectively. In the off-line simulation stage a first-approximation model coded in C language was used to describe the behavior of the dosage system. In the on-line simulation, this model was replaced by the actual plant connected via a National Instruments PCI-1200 data acquisition board to the PC running the software implementation of the fuzzy controller. Using these tools it was very easy to change the controller behavior (modifying the XFL specification file) and to record the system performance. Thus, different conditions were studied in order to select the better options for the hardware implementation. The defuzzification method chosen was the fuzzy mean so that the rule consequences are represented by singleton values. The final membership functions for the error input and the singleton values for the consequences were shown in Fig. 4.

The VHDL description of the fuzzy controller was obtained from its XFL specification with the aid of the *xfvhdl* tool of *Xfuzzy*. On the other side, the FPGA synthesis and implementation process were carried out with Synopsys *FPGA Express* and Xilinx *Foundation 4.1i*, respectively. Since the fuzzy inference engine implemented in the FPGA works as a coprocessor of the 8031 microcontroller, several interface circuits also had to be added into the FPGA. The timing task is implemented by one of the timers available in the microcontroller. Using 6 bits for input/output resolution, 5 bits for coding the membership degree, and memory-based MFCs, 98% of the FPGA CLBs were employed, and the maximum operational frequency was able to reach 20 MHz. However, the general clock frequency for both the FPGA and the microcontroller was lower than 20 MHz because the on-board 8031 highest operating frequency is 12 MHz. At a 10 MHz frequency, the fuzzy inference cycle takes 700 ns, which is shorter than the execution time of a single microcontroller instruction.

The software implementation in the 8031 was performed by using several development tools available for the MCS51 family. The code downloaded to the microcontroller covers the typical tasks of a control cycle (acquisition and pre-processing, communication with the inference engine, post-processing and output generation), and includes several interrupt service routines as well as routines to allow communication with a PC in order to evaluate the system performance.

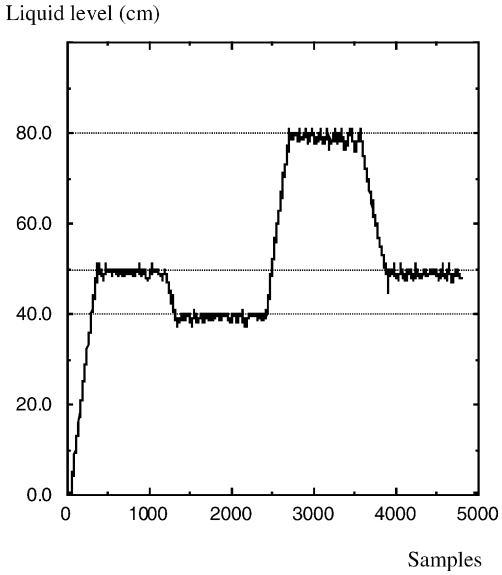


Fig. 10. Experimental results.

The controller thus designed has proved to be satisfactory in different experimental tests, providing a robust control and ensuring a good behavior of the dosage system even under hard changes in the aperture of the manually-controlled valves. As an example, Fig. 10 shows some of the experimental results obtained with this HW/SW fuzzy control system. It represents the evolution of the liquid level in one of the tanks versus the number of samples when the required level changed from 50 to 40 cm and then to 80 and 50 cm again. A sample was taken every 100 ms. The ripple shown in the figure is due mainly to having used only 6 bits in the hardware implementation of the system.

5. A fuzzy controller as a system on programmable chip

The above controller realization can be improved by including an embedded microcontroller together with

the fuzzy inference module into an FPGA. Thus, direct interfacing between both is possible while external connections are eliminated. This is an important advantage over the previous system in which the communication between the external microcontroller and the fuzzy inference module within the FPGA is carried out through external data buses and, hence, some 8031 resources can remain unexploited. Including the embedded microcontroller allows for complete use of all of its resources. Consequently, the fuzzy inference architecture can be easily reconfigured using the microcontroller I/O ports.

This is not the only advantage of using a SoPC approach. There are a lot of embedded cores of standard processors which run faster than the original ones, so an important speed increment can be obtained. This speed increment is due to advances in the processor architecture design. For instance, there are 8051 cores which can run up to twelve times faster than the standard by following a RISC approach and reducing the number of clocks per machine-cycle [22]. In addition, programmable device technology usually offers operating frequencies higher than classical microcontrollers. Therefore, the total speed increment can be significant.

Of course, a larger FPGA included in another development board is needed for a SoPC approach. We have employed the Digilab2E (D2E) FPGA-based development board, which is an adequate prototyping platform for low-cost moderate-to-complex SoPC [23]. The D2E board includes a medium-complex Spartan2E XC2S200E FPGA from Xilinx with 1176 CLBs. The board also includes a serial interface to accommodate a RS-232 standard port and a parallel interface for JTAG programming and parallel communication. A set of connectors allows the D2E board to be connected to external devices.

The size of the available FPGA allows us to tackle more complex control problems. In fact, the fuzzy inference module chosen as an example corresponds to a fuzzy-PID controller with three 8-bit inputs (five fuzzy sets each) and one 8-bit output. Membership functions for the FIM are shown in Fig. 11. Fuzzy mean was again used for defuzzification. Note that, unlike in the previous controller, where only 9 (3×3) rules were used to define the control strategy, the amount of rules grows now to 125 ($5 \times 5 \times 5$). Thus, the approaches employed to store the knowledge base deserve a more detailed discussion.

The functionality of a fuzzy system is determined by the contents of its knowledge base (membership functions used in antecedents and consequences, and fuzzy rules). From the point of view of implementation, both the antecedent and the rule memories can be implemented by different approaches: using RAM or ROM memory (depending on if it is necessary or not to modify the system knowledge base concurrently with its operation) or with combinational circuits (which allows a reduction of the size of the logic).

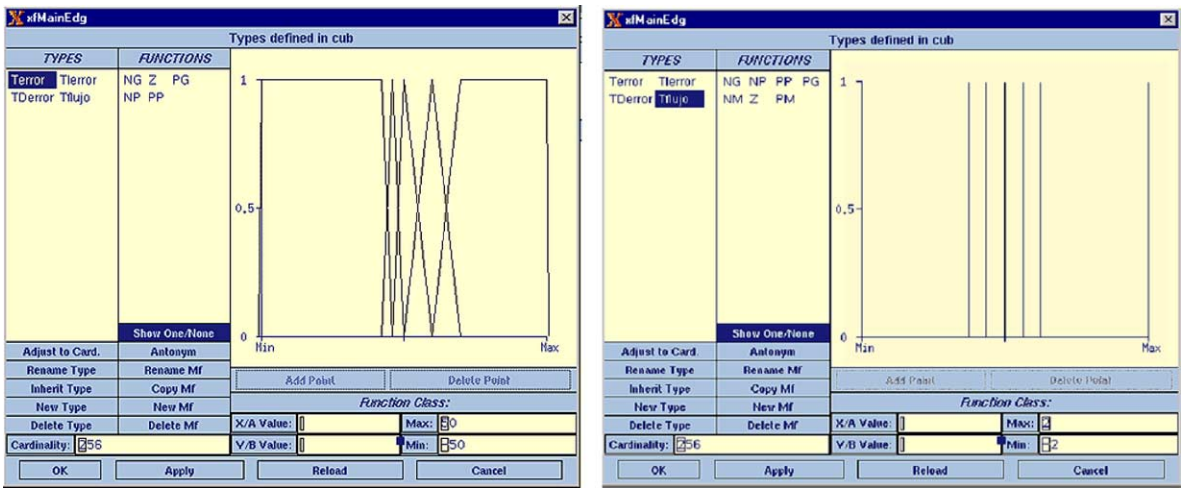


Fig. 11. Membership functions for the Fuzzy-PID controller.

Many FPGA devices store the internal configuration in RAM, thus allowing us to change the circuit functionality by reprogramming the FPGA. This characteristic makes it possible to actualize the knowledge base of the fuzzy system even when it has been implemented with ROM or combinational logic. The availability of automatic CAD tools which allow for a quick obtaining of the FPGA configuration file from the high-level system specifications eases this process greatly.

When the change in the knowledge base must be realized concurrently with the system operation (for instance, in adaptive systems) it is necessary to use RAM memory. The basic building blocks of certain FPGAs, such as those provided by Xilinx, are made up of small memories working as look-up tables (LUTs) able to implement any logic function of a certain number of inputs. These elements can be configured to form distributed RAM blocks which can be used by the designer. Additionally, some FPGA devices, such as the Spartan2E offer new implementation alternatives because they incorporate specific block RAM memories (BRAM) which admit different configurations. Using BRAM for the memories frees CLB resources and more logic can be included.

In general, the use of BRAM to accommodate the rule memory may contribute to a great area reduction if there are a lot of rules (i.e. there are a lot of inputs and/or many input fuzzy sets). In the case of MFCs the use of BRAM may be useful in order to avoid the high area consumption problems related to the exponential growth of memory-based MFCs. For instance, the memory-based MFCs required in the application example of the previous section are 64×12 bits each, but when the input resolution is increased to 8 instead of 6 bits, the memory capacity required is quadruplicated and may consume a great FPGA area. Therefore, the use of BRAM in order to implement memory antecedents is a very attractive option. However, although arithmetic-based MFCs also need memory, as a consequence of its spatial structure, the BRAM implementation of this memory is not recommended due to its inherent inefficient utilization. For instance, if the input membership functions required in the application example of the previous section are provided by arithmetic-based MFCs (using also 6 bits for the slope codification), the parameter-memory capacity PM would be 4×24 bits and the implementation of this very simple memory would need two BRAM in a Spartan2E which would be highly under used.

Table 1
Comparative implementation results (in % CLBs) among different architecture options

MFC	Block RAM (%)	Distributed RAM (%)	ROM (%)	Logic (%)
Memory-based	9	80	34	29
Arithmetic	14	33	17	17

Table 1 shows the Spartan2E200 FPGA area consumption (in CLBs percent) for different implementation options of the fuzzy inference module required for the application example. The columns reflect the implementation options whereas the rows show the type of MFC used. For memory-based MFCs, the "*Block RAM*" option leads to the lowest CLB consumption. In this case, seven BRAM are needed: one for the rule memory and two for each 256×18 antecedent memory. The "*Distributed RAM*" option has the highest area due to the intensive use of CLBs to implement the antecedent and rule memories. Its use makes no sense unless a programmable knowledge base is required. The last two columns show the area occupation when the contents of the database are fixed at implementation time: the "*ROM*" column refers to a ROM extraction done by XST whereas the "*Logic*" column refers to a ROM implementation as distributed logic. The 5% difference between both is a consequence of the logic reduction which can be carried out in the latter case. The second row of Table 1 shows that all the options are very similar when arithmetic-based MFCs are used. This is due to the reduced size of the memory needed to store the membership function parameters. One BRAM is needed again to store the rule memory and three more are used for storing antecedent parameters (a 5×48 memory structure is required) when this option is chosen. Note again the BRAM resources which are wasted by implementing this memory. The 5% increase over the same option with memory-based MFCs is due to the arithmetic circuits needed to implement the MFCs.

As a result, a more complex fuzzy controller, with a higher resolution than the one presented in Section 4 occupies a small portion on the Spartan2E200 of a Digilab2E board (note that the one described in Section 4 occupied 98% of the FPGA),

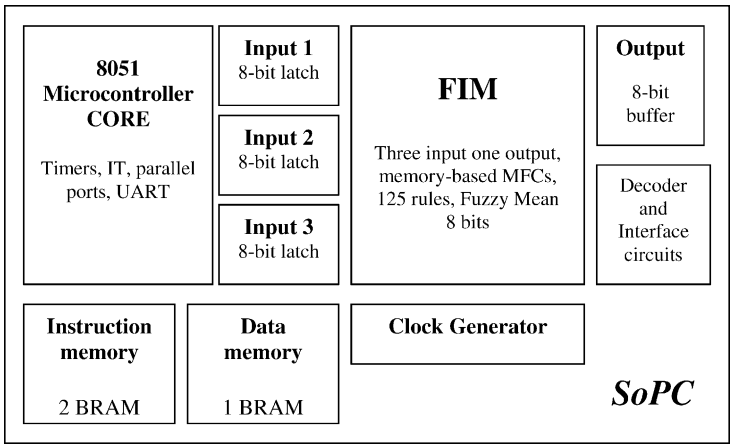


Fig. 12. Internal structure of the fuzzy controller as SoPC.

and, hence, there are enough resources to embed a general-purpose microcontroller. The 8051 microcontroller core used has been a free one from Oregano Systems (http://www.oregano.at) which is far away (in area and speed) from other licensed cores. This microcontroller is basically equivalent to the 8031 used in the first prototype (the 8031 microcontroller is actually an 8051 without internal ROM). The system also integrates the interface circuits between the microcontroller and the fuzzy module. Although the microcontroller core is not an optimum one, the implementation reports a 78% FPGA area utilization using the BRAM memory-based option and an 86% using an arithmetic MFC implemented as ROM option, from which 69% corresponds to the microcontroller core.

Additional memories have to be considered in the codesign of a fuzzy controller as a SoPC. The microcontroller core needs its proper internal data and code memory, so that some BRAM (organized as 512×8 bits) can be used to accommodate both memories. Internal data memory lies in only one BRAM whereas the number of BRAM for code memory depends on the final code size. The whole control program lies in two BRAMs of the Spartan2E200. Fig. 12 shows the internal structure of the final control system.

These results not only show that SoPC is a good choice for codesigning fuzzy controllers but open an interesting way of expanding the fuzzy inference architecture in order to achieve software-controlled configurability for the fuzzy module as well as more complex hierarchical structures.

6. Conclusions

The use of codesign techniques according to a given partitioning of the tasks assigned to the hardware and software components has proved to be an efficient strategy for designing high-speed and low-consumption fuzzy logic-based control systems. The ‘a priori’ partitioning presented here leads to a hardware implementation of the fuzzy inference module using an active-rule driven architecture and simplified defuzzification methods whereas the other tasks are implemented in software.

According to this strategy, a complete codesign flow for fuzzy controllers based on programmable platforms and using several CAD tools from the *Xfuzzy* environment has been described. Two implementation approaches have been presented. The first one using an external microcontroller and a medium complex FPGA shows its success through an actual application to solve a dosage problem.

The availability of powerful FPGAs, intellectual properties modules, and CAD tools, together with reuse techniques made it possible to develop the fuzzy controller as a whole system on a programmable chip. A SoPC implementation based on a Xilinx Spartan2E200 FPGA is the second approach exposed. The memory resources of the Spartan2E200 together with the fuzzy inference architecture options allow for different implementations in order to reduce CLB consumption. A fuzzy controller which includes a non-optimal 8051 microcontroller core, an 8-bit fuzzy

inference module, and the required interface circuits occupies 83% of the FPGA slice resources.

These results lead us to be optimistic about the advantages of codesigning more complex configurable and hierarchical fuzzy controllers as SoPCs.

Acknowledgements

This work has been partially supported by the Spanish CICYT Project TIC2001-1726.

References

[1] K.M. Passino, S. Yurkovich, Fuzzy Control, Addison-Wesley, Reading, MA, 1998.

[2] T. Terano, K. Asai, M. Sugeno (Eds.), Applied Fuzzy Systems, Academic Press, New York, 1994.

[3] J. Yen, R. Langari, L.A. Zadeh (Eds.), Industrial Applications of Fuzzy Logic and Intelligent Systems, IEEE Press, New York, 1995.

[4] L. Reznik, Fuzzy Controllers, Newness, London, 1997.

[5] I. Baturone, A. Barriga, S. Sánchez-Solano, C.J. Jiménez, D. López, Microelectronic Design of Fuzzy Logic-Based Systems, CRC Press, Boca Raton, 2000.

[6] B. Tabbara, A. Tabbara, A. Sangiovanni, Function/ Architecture Optimization and Co-Design of Embedded Systems, Kluwer Academic Publishers, Dordrecht, 2000.

[7] V. Salapura, A Fuzzy RISC Processor, IEEE Transactions on Fuzzy Systems, vol. 8, No. 6, December 2000.

[8] A.P. Ungering, K. Goser, Architecture of a 64-bit fuzzy inference processor, in: Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZIEEE'94, Orlando, 1994, pp. 1776–1780.

[9] C. Von Altrock, Adapting Existing Hardware for Fuzzy Computation, Institute of Physics Publishing, 1998.

[10] M.J. Patyra, E. Braun, Fuzzy/Scalar RISC processor: architectural level design and modelling, in: Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZIEEE'96, New Orleans, 1996, pp. 1937–1943.

[11] Y.D. Kim, H. Lee-Kwang, High speed flexible fuzzy hardware for fuzzy information processing, IEEE Trans. Syst. Man Cybernetics 27 (1) (1997) 45–55.

[12] A. Pagni, Digital approaches, in: Handbook of Fuzzy Computation, Institute of Physics Publishing, 1998.

[13] D.R. López, C.J. Jiménez, I. Baturone, A. Barriga, S. Sánchez-Solano, Xfuzzy: a design environment for fuzzy systems, in: Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZIEEE'98, Anchorage, 1998, pp. 1060–1065.

[14] S. Sánchez-Solano, R. Senhadji, A. Cabrera, I. Baturone, C.J. Jiménez, A. Barriga, Prototyping of fuzzy logic-based controllers using standard FPGA development boards, in: Proceedings of the 13th IEEE International Workshop on Rapid System Prototyping, RSP'02, Darmstadt, 1–3 July 2002, pp. 25–33.

[15] A. Cabrera, S. Sánchez-Solano, R. Senhadji, A. Barriga, C.J. Jiménez, Hardware/software codesign methodology for fuzzy controllers implementation, in: Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZIEEE'2002, Honolulu, 12–17 May 2002.

[16] D.R. López, F.J. Moreno, A. Barriga, S. Sánchez-Solano, XFL: a language for the definition of fuzzy systems, in: Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZIEEE'97, Barcelona, 1997, pp. 1581–1591.

[17] R. Senhadji, S. Sánchez-Solano, D.R. López, A. Barriga, Xflab: an on-line verification tool for fuzzy controllers, in: Proceedings of the IPMU'2000, Madrid, 2000, pp. 44–49.

[18] E. Lago, C.J. Jiménez, D.R. López, S. Sánchez-Solano, A. Barriga, Xfvhdl: a tool for the synthesis of fuzzy logic controllers, in: Proceedings of the DATE'98, Paris, 1998, pp. 102–107.

[19] S. Sánchez-Solano, A. Barriga, C.J. Jiménez, J.L. Huertas, Design and applications of digital fuzzy controllers, in: Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZIEEE'97, Barcelona, 1997, pp. 869–874.

[20] H. Hellendoorn, C. Thomas, Defuzzification in fuzzy controllers, J. Intelligent Fuzzy Syst. 1 (1993) 109–123.

[21] XS40, XSP Board V1.4, User Manual, XESS Corporation, USA, 1999.

[22] CAST 8051 Core Family: An Overview of Product Family and Performance, CAST Inc., September 2002.

[23] Digilab 2E Reference Manual, Digilent Inc., 2002.