Hindawi Publishing Corporation
Advances in Fuzzy Systems
Volume 2012, Article ID 475894, 11 pages
doi:10.1155/2012/475894

Research Article

Designing High-Performance Fuzzy Controllers Combining IP Cores and Soft Processors

Oscar Montiel-Ross, Jorge Quiñones, and Roberto Sepúlveda

Instituto Politécnico Nacional, CITEDI, Avenida del Parque 1310, 22510 Tijuana, B.C., Mexico

Correspondence should be addressed to Oscar Montiel-Ross, o.montiel@ieee.org

Received 5 May 2012; Accepted 3 June 2012

Academic Editor: Oscar Castillo

This paper presents a methodology to integrate a fuzzy coprocessor described in VHDL (VHSIC Hardware Description Language) to a soft processor embedded into an FPGA, which increases the throughput of the whole system, since the controller uses parallelism at the circuitry level for high-speed-demanding applications, the rest of the application can be written in C/C++. We used the ARM 32-bit soft processor, which allows sequential and parallel programming. The FLC coprocessor incorporates a tuning method that allows to manipulate the system response. We show experimental results using a fuzzy PD+I controller as the embedded coprocessor.

1. Introduction

Nowadays, the term System on Chip (SoC) gains terrain since the trend towards the use of a highly efficient hardware platform for real-time processing is increasing [1]. Modern FPGA devices that allow to mix digital and analog signal makes them a good choice to use them for SoC design [2].

FPGA platforms are designed to achieve higher integration levels in low-power low-cost electronic systems by embedding processors with efficient architectures such as DSP, RISC, multicore processor systems, buses, on-chip memory blocks, peripheral devices [3]. FPGA-based system architectures support the combination of user defined synthesizable Intellectual Property (IP) blocks with IP blocks and software drivers and libraries provided by the manufacturer [4]. The use of SoC allows total independence of a desktop computer system, reducing costs, and increasing performance of applications, avoiding communication interface bottlenecks, latency, and data loss [5].

The SoC design based on modern FPGAs may incorporate dedicated processors embedded in the silicon known as "Hard", and programmable processors known as "Soft" that are implemented in the programmable logic resources of the FPGA, or a mixed of both.

The real-world problems are diverse and complex, treating many of them becomes difficult, because they make us face many scientific and technological barriers, such as, mathematical modeling and high-speed processing for data processing and control applications [6].

For control applications, there exist many techniques and strategies to make a system behaves according to a plan; for example, to use a Fuzzy Logic Controller (FLC) that is considered as a control strategy based on rules, which are usually raised by the knowledge from an expert. This may be crucial in control problems that could present difficulties in constructing accurate mathematical models [7, 8].

There are two typical options to carry out applications of FLC embedded into an FPGA.

(1) To describe the FLC in C language and then use a specialized tool (compiler) to translate Handel C to bitstream [9, 10].

(2) To use VHDL to describe the FLC, this can be used as (a) standalone controller, (b) incorporated to a hard/soft processor as a Soft Core connected to the system bus, (c) incorporated to a hard/soft processor through an internal input/output interface [11, 12].

The complexity of modern systems requires more design efforts by increasing developing costs; so, to minimize them, the reutilization of already tested circuits is a necessity. Any functional component to be reused in the form of

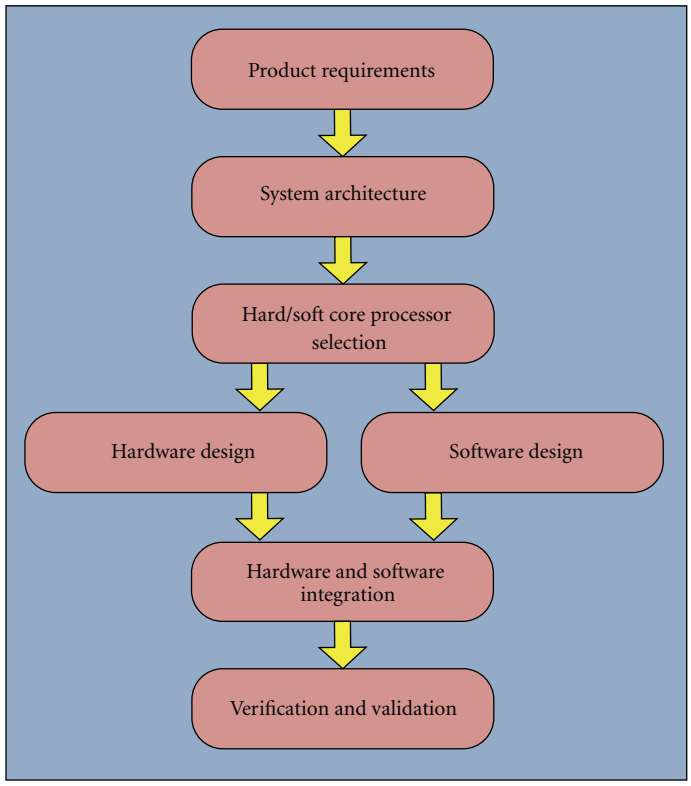


Figure 1: Design process of an embedded system.

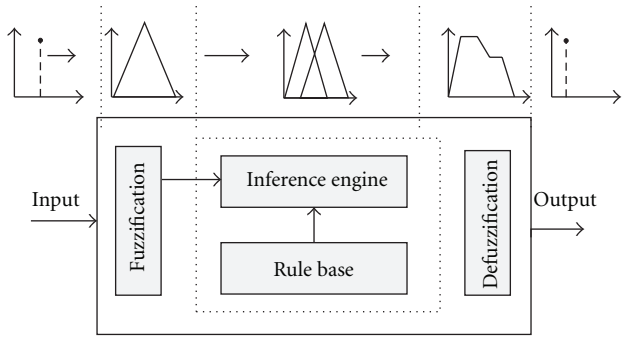


Figure 2: Fuzzy inference system.

an already-designed electronic component, or fabricated hardware constitute an Intellectual Property Core (IP-Core).

This paper presents a methodology to integrate a FLC to a SoC through an input/output internal port; the idea is to use the FLC as a coprocessor. Furthermore, an experimental study of high-performance computing using fuzzy modeling implemented in FPGA-based systems is presented.

Other studies that have addressed the same subject with a different focus are. In [13], the authors describe their experiences designing real-time hardware/software for SoC and they addressed the problem of data acquisition for the ANTARES neutrino experiment, and for the problem of a selective read-out processor for an electromagnetic calorimeter. In [14, 15], the authors present the architecture and VHDL code to implement a type-1 FLC, while in [16] the architecture and VHDL code to implement a type-2 FLC and experiments are presented.

In literature, there exist many interesting works that deal with fuzzy controllers that can be embedded into an FPGA, for example. [17] presents the genetic optimization of Membership Functions (MFs) for an Incremental Fuzzy PD Controller, [18] shows the optimization of MFs to regulate a servomechanism with backlash.

The organization of this paper is as follows. Section 2 explains the design process of an embedded system, how to integrate a fuzzy coprocessor into a SoC, and the debugging process. In Section 3, the problem formulation, experimental plant, control objective, and electromechanical limitations concerning time are given. In Section 4, the generic architecture that integrates the ARM processor and the FLC as Intellectual Property cores (IP core) is described. In Section 5, the main characteristics of the FLC are described, and the methodology to tune the FLC using one variable is explained. In Section 6, the experiments' sets and results are explained. Finally, in Section 7, the conclusions of this work are given.

2. Integrating a Fuzzy Coprocessor in a SoC

Figure 1 illustrates the design process of a SoC-embedded system. It is a standalone dedicated hardware computer with custom peripherals incorporated to the system as IP cores, and specialized software to solve a specific problem.

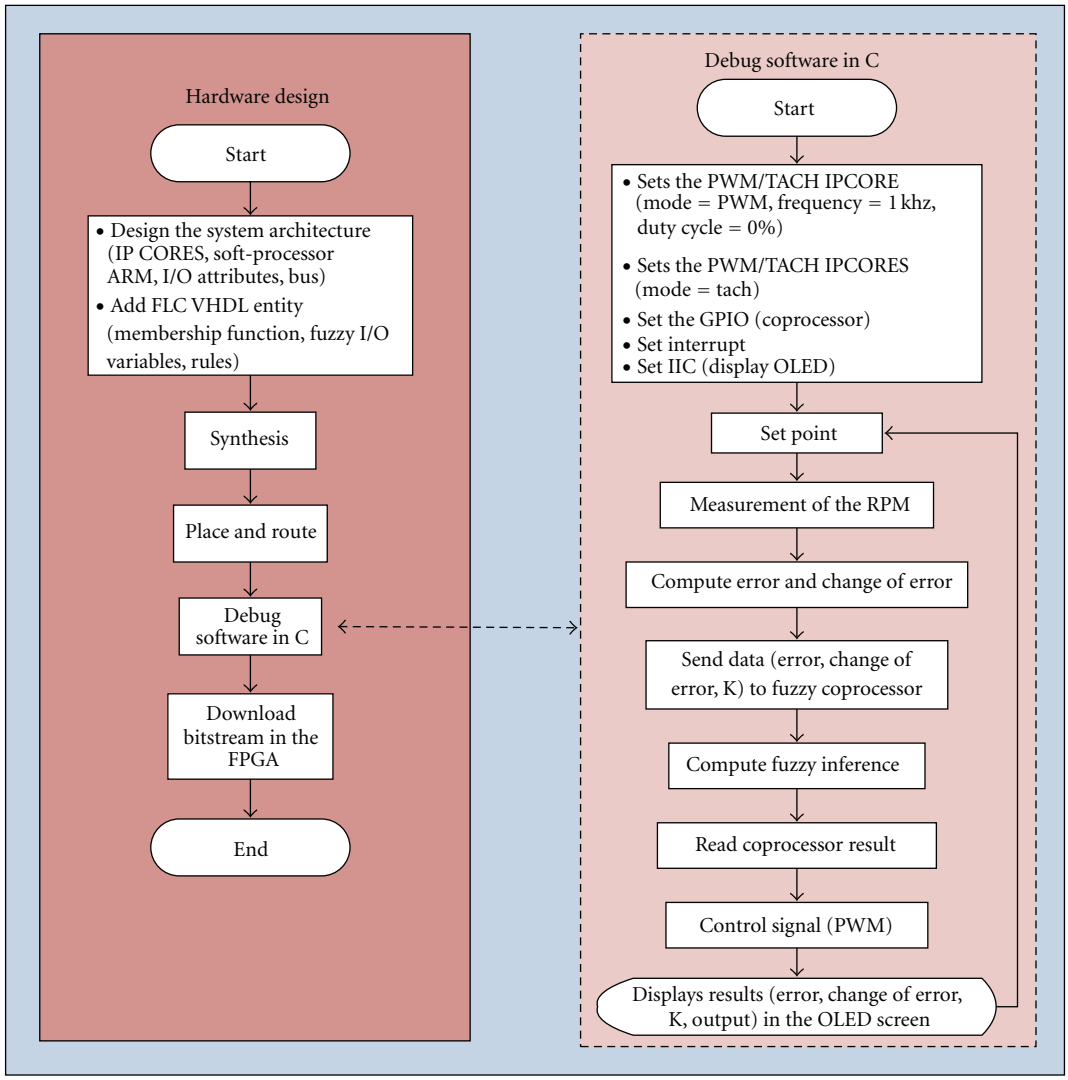


Figure 3: Flow diagram of the debugging process of a C program executed in the FPGA base system.

2.1. Designing the FLC as IP Core. Figure 2 shows the typical way of representing a FLC; it is composed of three stages: the fuzzification stage, the inference engine that contains the rule base and database, and defuzzification stage [19].

In general, the steps to incorporate into an FPGA Fusion an FLC as Soft IP core using the Libero software are.

(1) Create the Entity Design of the FLC by using either a Hardware Description Language (HDL), structural schematic, or mixed-mode (schematic and Register Transfer Logic "RTL"). We used VHDL (Very High Speed Integrated Circuit HDL) to specify all the fuzzy engine and parameters.

(2) Test the functionality of the FLC Entity Design using a test bench program such as the Model Sim for VHDL.

(3) Create the soft processor and incorporate the FLC IP coprocessor [20] entity to the system bus trough the GPIO IP.

(4) Edit and debug the ARM firmware in C language to complement the controller's design incorporating the reading and conditioning of the process variables.

(5) Download the bitstream to the target FPGA development board.

The obtained FLC is a portable, reusable, and nonencrypted Soft IP core. It was designed to be used through an input/output interface, which is an advantage because many of the commercial IP cores have the inconvenient that are not compatible with all the system buses.

Figure 3 illustrates the hardware design and debugging of the software written in C language.

3. Problem Formulation

The main target of this work is to give a methodology to integrate a soft IP FLC to a SoC in order to develop high-performance applications; whereas, the experiments

Figure 4: Inverted pendulum system.

are focused to demonstrate the advantages of using high-performance FLC for real-life problems. To achieve the objective paper, we are including two well-known problems.

In the first problem, we used a plant with the next main components: a Pittman DC geared-motor model GM9236S025-R1, a ± 12 V_{DC} power Supply, an H-bridge board, and a PWM system to provide the motor with the necessary average power to reach the desired speed with and without load, and disturbances. Therefore, the *control objective* is to reach the wanted speed as fast as possible with the minimal overshoot, which is achieved by the correct calculation of the relation T_{ON}/T_{OFF} to fulfill the target.

Two important characteristics of this motor are the electrical time constant τ_E and the mechanical time constant τ_M whose values are 1.06 ms and 8.5 ms, respectively; so, we can consider global response time constant of 9.16 ms. This time is significant since it is the time required for the motor's speed to attain 63.2% of its final value for a fixed voltage level.

The DC motor alone is a linear plant since the no-load speed is directly proportional to the DC supply voltage. However, the system becomes nonlinear when a switching supply is applied, in this case by using PWM; in addition, the application of load and disturbances increases nonlinearities.

In the second problem, the plant is an inverted pendulum; see Figure 4. The cart is mounted over an aluminum frame; it is moved from one side to the other using a ball screw controlled by a motor. The pole balancing is mounted over the cart and coupled to a quadrature optical encoder Model 121, which provides 300 counts per revolution (CPR).

4. SoC General Description

In Figure 5, the designed system architecture for FPGA Fusion [21] of the Actel company is shown, embedded into the FPGA are. The ARM processor, two memory blocks, a general-purpose input output (GPIO) interface, timers, interrupt controller (IRQ), IIC, serial port (UART), pulse width modulator/tachometer block, and the FLC block [14, 15]; all the embedded components are IP cores. External to the FPGA are a DC motor with a high-resolution quadrature optical encoder, the plant's power supply, an H-bridge for power control, a personal computer, and a digital display.

The FPGA Fusion allows to incorporate the soft processor ARM cortex, as well as other IP Cores (IPCORE) to make a custom configuration. The ARM cortex handles a 32-bit bus for peripheral control named Advanced Peripheral Bus (APB). Note in Figure 5 that the block FLC contains the design of the fuzzy controller integrated to the system as a soft IP CORE, similarly the rest of the IPCOREs are general purpose devices from the catalog of cores provided by Actel and the system board seller. In this system, the FLC IP core has the advantage that it is not dependable of the bus system, and it provides to the user the capacity of handling very easy without the need of knowing the internal functionality; on the other hand, because it is not encrypted, the content can be read.

Figure 5 shows how to connect the FLC IP core to the ARM processor through the GPIO. The FLC has seven inputs and two output. The inputs are "clk", "ce", "rst", "k", "error", "c.error", and "w". The input "clk" is a 50 Mhz system clock, the aim of the "ce" input is to enable or disable the FLC, and the input "rst" restores all the internal registers of the FLC, the input "w" working together with "ce" allows to start a fuzzy inference cycle; all they are one-bit size. The four-bit input "k" is for modifying the support of the input membership functions to change the system response. The eight-bit input "Error" and Change of Error "c.error" are the controller inputs. The outputs are "Out" and "IRQ". "Out" is eight-bits wide and it is the crisp output value. "IRQ" is the interrupt request FLC output. The system has five membership functions (MFs) for each input and output. The input and output MFs are trapezoidal at the extremes, and the remaining are triangular. This FLC uses the simple tuning algorithm for modifying the system's response in an intuitive way [14, 15]. Figure 6 shows the FLC entity and GPIO IP bus connection.

The GPIO IP has two 32-bit wide ports; one for output (write bus) and one for input (reading bus). The output bus connects the GPIO IP to the ARM cortex using the 32-bit bus APB. The input bus connects the FLC IP to the GPIO IP. In this configuration, the FLC works as a coprocessor of the ARM cortex processor.

The ARM processor writes a "k(3:0)" value for setting up the input MFs, it provides the Error and Change of error values for the "Error(7:0)" and "c.error(7:0)" FLC

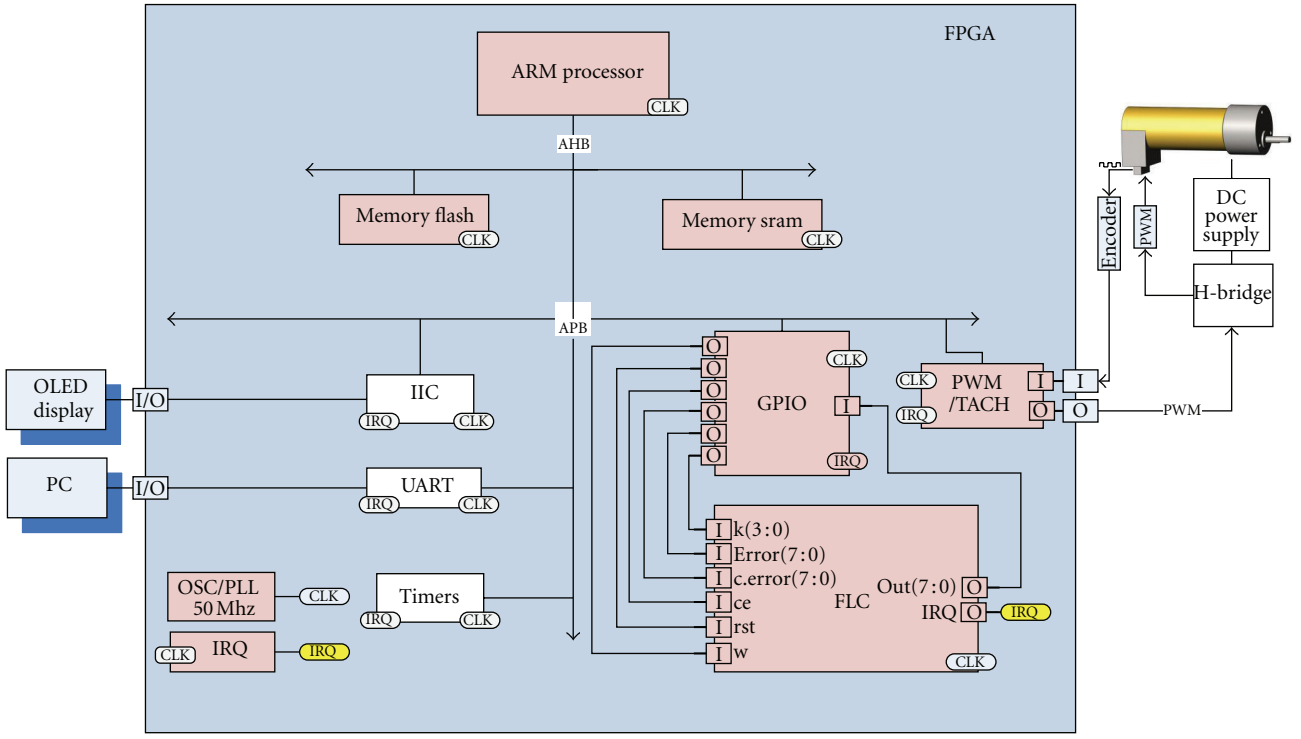
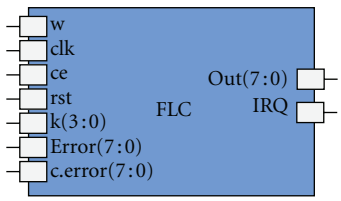


Figure 5: Overview of the general system.



Output bus																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Not connected									w	ce	rst	c.error								Error								k			

Input bus																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ground																								Out							

Figure 6: FLC entity and GPIO bus connection.

inputs, this is achieved using the output variables k[3:0], Error[11:4], and c.error [19:12] of the GPIO IP. Similarly, the ARM reads the FLC output "Out(7:0)" through the GPIO IP using the 8 bit input variable Out(0:7).

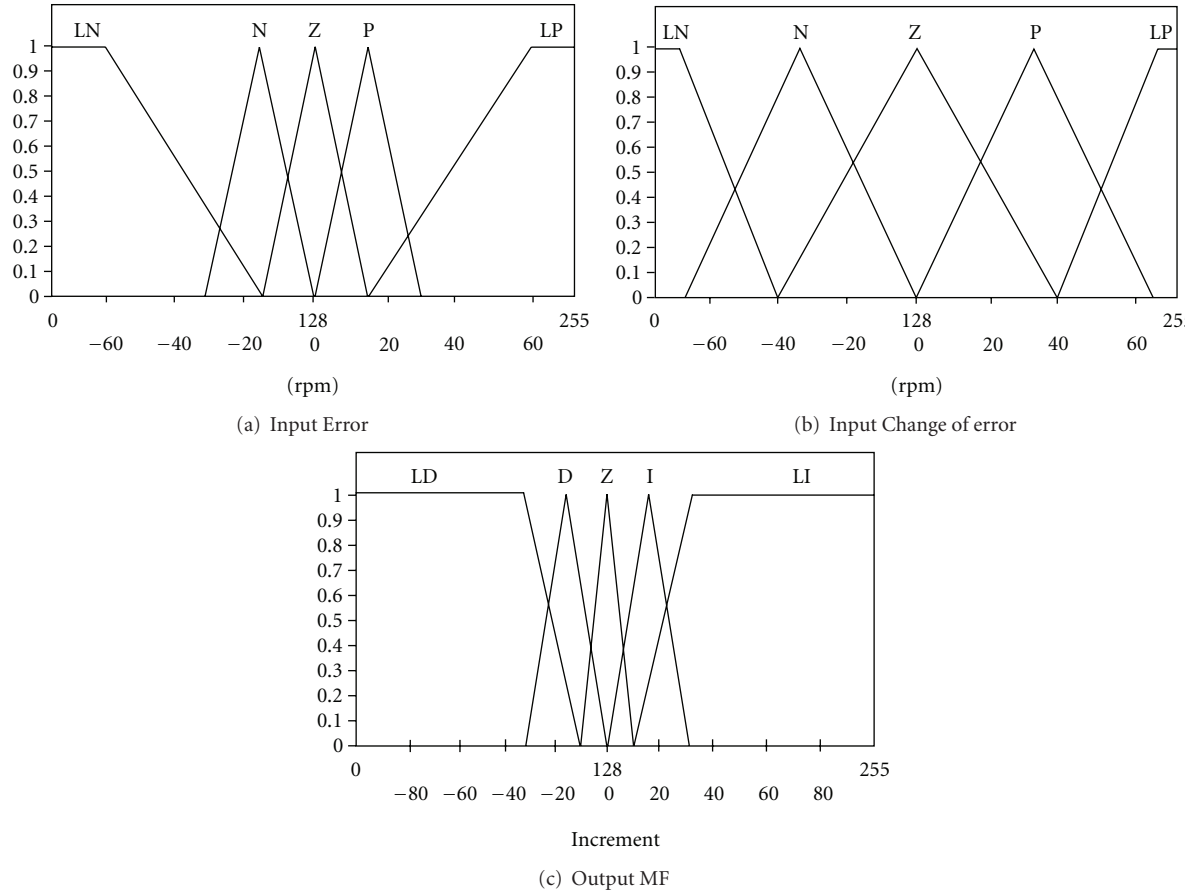
5. FLC Characteristics

For each input of the FLC, Error and Change of error, we defined five MFs: LN (large negative), N (negative), Z (zero), P (positive), and LP (large positive). The universe of discourse for these membership functions is in the range $[-80, 80]$.

For the output of the FLC, we have five MFs: LD (large decrement), D (decrement), Z (zero), I (increment) and LI (large increment), with the universe of discourse in the interval $[-100, 100]$.

Figure 7 shows the input/output membership functions, and the rule matrix of the FLC, integrated by 25 rules, is shown in Figure 8.

5.1. Tuning the Controller. The FLC IP has a three-bit input to manipulate the MFs to improve the desired response by using the method referred in literature as Simple Tuning Algorithm (STA) [22, 23]. There are different methods to tune a FLC, for example, evolutionary computation, Fuzzy Knowledge Base Controllers, and so forth, where the aim is to search the optimal solution in base of objective function, gradient error, and others but in most of the cases these



(a) Input Error

(b) Input Change of error

(c) Output MF

Figure 7: The FLC has two inputs: Error and Change of error, and one output. The output provides a value of increment/decrement to be use by the integral part of the PD+I controller. The first scale $[0, 255]$ are the real MFs bit values definition in the FLC. The second scale $[-80, 80]$ and $[-100, 100]$ are the operation values for this application. The scale conversion is achieved by the ARM processor.

ce \ e	LN	N	Z	P	LP
LN	LI	LI	I	D	LD
N	LI	I	Z	Z	LD
Z	LI	I	Z	D	LD
P	LI	Z	Z	D	LD
LP	LI	I	D	LD	LD

Figure 8: Rule matrix of the FLC.

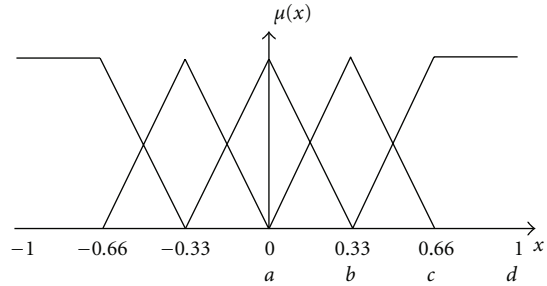


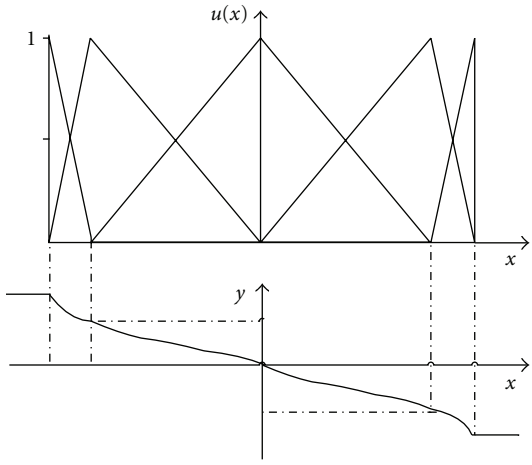
Figure 9: Normalization of the input MFs. The tuning factor is $r = 1$.

methodologies have convergence and mathematical representation problems; furthermore, often they require many system resources and have considerable big computational complexity for online adaptation of real time systems.

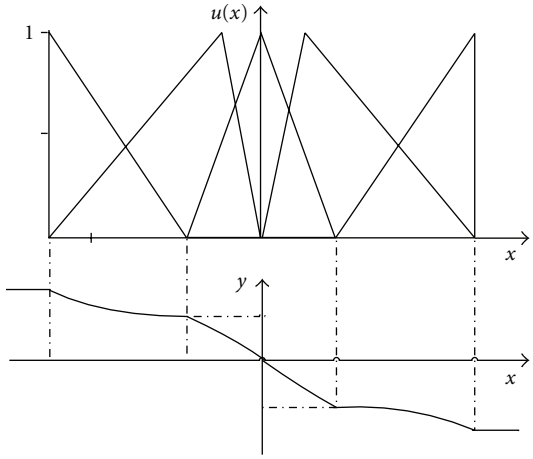
The STA takes advantage of the typical characteristic of error behavior around the set point, so it is possible to use a predefined set of MFs and rules and modify their support according to a very simple arithmetic expression. In this example, the two inputs were modified using only a tuning factor [23]; however, it is likely to modify both inputs using two tuning factors [22]. In Figure 9, the normalized inputs are shown; Figures 10(a) and 10(b) show how the MFs are modified by the application of the STA algorithm and their effect in the system behavior.

Considering all the points in the universe of discourse that defines every MF of each input, a vector named $V_{\mathrm{OP_{initial}}}$, is codified. The basic idea of the STA is to change this vector using an exponential function $r(k)$ named the tuning factor such as $V_{\mathrm{OP_{final}}} = (V_{\mathrm{OP_{initial}}})^{r(k)}$, that produces the changes to the system which is shown in Figure 10. This leads us to analyze the next three cases using Figure 9:

(1) $r = 1$: this is the case where the normalized MFs have no change; that is, $a - b = b - c = c - d$,



(a) The tuning factor is $r < 1$



(b) The tuning factor is $r > 1$

FIGURE 10: Modifying the input MFs using the formula $V_{\mathrm{OP}_{\mathrm{final}}} = (V_{\mathrm{OP}_{\mathrm{initial}}})^{r(k)}$.

(2) $r < 1$: after the application of the tuning factor, the distances between the middle MFs operation points are larger than the external; so, $a^r - b^r > c^r - d^r$, producing the expansion of the MFs, as illustrated in Figure 10(a).

(3) $r > 1$: the opposite case of previous, after the application of the tuning factor, the distances between the two external MFs are larger than the others; that is, $a^r - b^r < c^r - d^r$. This produces the compression of the MFs, this effect is shown in Figure 10(b).

Therefore, the STA consists basically in four steps.

(1) *Tuning Factor Selection.* A number $k \in [0, 1]$ to define the tuning adjustment level is used. $k = 0$ is the biggest time, and $k = 1$ the smallest.

(2) *Normalization of the Ranges of the Fuzzy Controller's variables.* The range of each input fuzzy variable is modified in order to have the lower and upper limits equal to -1 and $+1$, respectively, see Figure 9.

(3) *Tuning Factor Processing.* Once the range is normalized, the new vector of operation points will be given by.

$$V_{\mathrm{OP}_{\mathrm{final}}} = (V_{\mathrm{OP}_{\mathrm{initial}}})^{r(k)}, \tag{1}$$

where $V_{\mathrm{OP}_{\mathrm{initial}}}$ are the normalized values of the MFs in the x-axis and $r(k)$ can be one of the following polynomials:

(a) The value $r \in [1/40, 3]$

$$r(k) = \frac{30k^3 + 37k^2 + 52k + 1}{40}, \tag{2}$$

(b) The value $r \in [0, 4]$

$$q(k) = 4k^2. \tag{3}$$

Both polynomials can be implemented into an FPGA, the first one needs more calculation, whereas the second option offers reduction of the polynomial size, reducing computational cost. However, using a lookup table the size is not a problem.

(4) *Renormalization of the Ranges of the Fuzzy Variables.* Convert the normalized range to the previous range of the system.

The method can be applied to both inputs, as it was shown in [22, 23]; in the first work, two different tune factors were used, one for each fuzzy input. In the second work, experiments modifying both inputs using the same tuning value were shown. Results of both methods are satisfactory, using [23, 24] requires fewer resources and is easier to tune the system using just one variable.

6. Experiments and Results

With the aim of evaluating equivalent implementations of high-performance FLC embedded into an FPGA; two different systems were tested.

6.1. System 1: Speed Control of a DC Motor. For system 1, two sets of experiments were conducted. The first set comprises six experiments using different development platforms where the FLC was implemented to solve the previously regulation of speed problem. In the second set, the configuration of ARM soft processor with the FLC IP was chosen, because it demonstrated to be the best for the target of this work; therefore, we tested the controller using the STA to tune the controller.

For All Experiments. We consider a complete fuzzy inference, the process of fuzzification of crisp data, to infer a conclusion using the Mamdani inference engine, and defuzzification to obtain a crisp value.

Next, for each experiment, particular characteristics are described. Table 1 summarized the results.

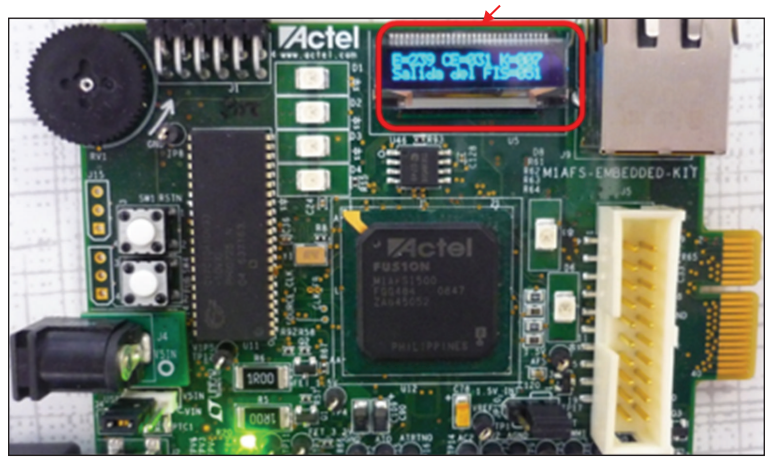


Figure 11: Actel development board for FPGA model "Embedded M1AFs1500 KIT."

Table 1: Comparing performance of FLC implementation for different development platforms.

Experiment	Platform	CPU	Clock frequency	Runtime (ms)
1	PC	Core 2 duo	2.66 GHz	20
2	Spartan 3	Microblaze	50 MHz	37
3	Virtex 5	Microblaze	100 MHz	16
4	Virtex 5	Power PC	100 MHz	12
5	Atmel AVR	8-bit Microcontroller	16 MHz	87
6	Fusion	ARM with fuzzy coprocessor	50 MHz	0.000160

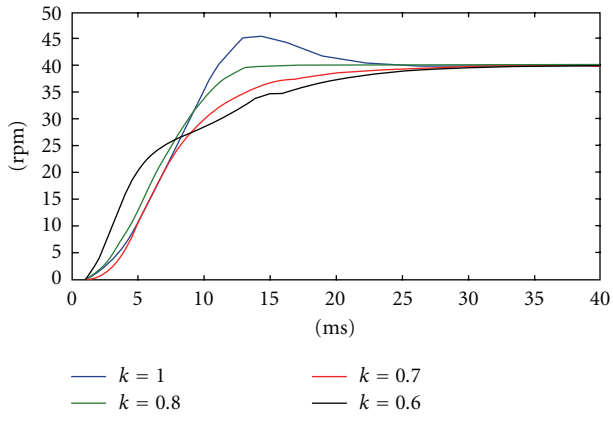


Figure 12: System response for different tuning factors.

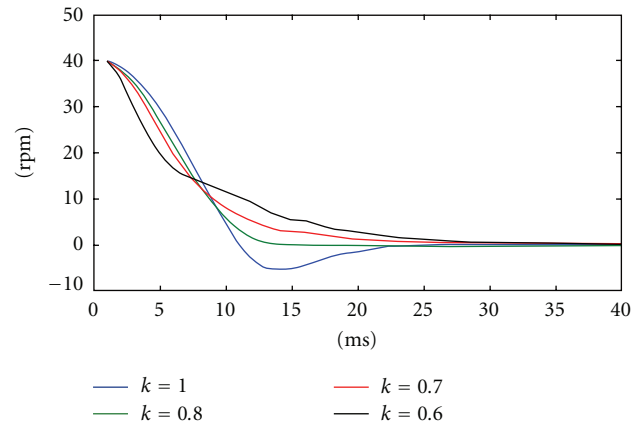


Figure 13: Plot of errors in the system response produced by different k tuning values. Note that $k = 0.8$ produces less errors.

6.1.1. Set 1 of Experiments

Experiment 1. The PC system is based on an Intel Core 2 Duo, with 6 GB of RAM. The FLC was implemented using the Fuzzy Logic Toolbox from Matlab-Simulink. The FLC writes and read information of the control plant using serial communication. A complete inference last 20 ms. We did not consider in the calculus the time due to serial communication.

Experiment 2. We used the Spartan 3 FPGA mounted in the Starter kit of Xilinx. We implemented into the FPGA the 32-bit Microblaze soft-core with 1 MB of RAM; the system clock is 50 Mhz. The FLC was implemented using C language. As the operating system we used the kernel of Xilinx standalone.

Experiment 3. In this test, a Virtex 5 FPGA mounted in the experimental board "ML507 FPGA technology" from Xilinx was used to embed a Microblaze soft-processor system, with 16 MB of RAM, and 2 KB of cache memory; the system clock is 100 Mhz. The FLC was implemented using

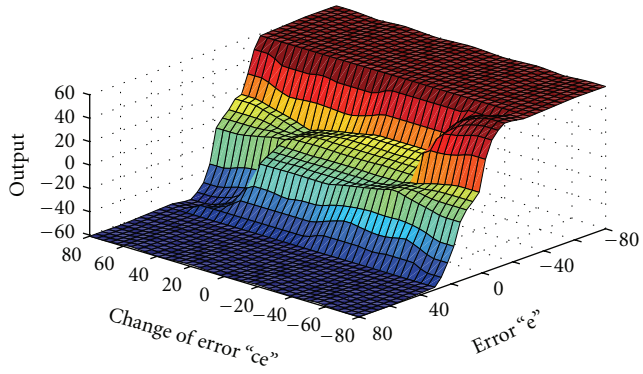


Figure 14: Surface control for a tuning factor $k = 0.5$.

ce \ e	LN	N	Z	P	LP
LN	LL	LL	LL	L	R
N	LL	L	L	Z	R
Z	LL	L	Z	R	RL
P	L	Z	R	R	RL
LP	LL	R	RL	RL	RL

Figure 15: Rule matrix for the inverted pendulum on a cart system.

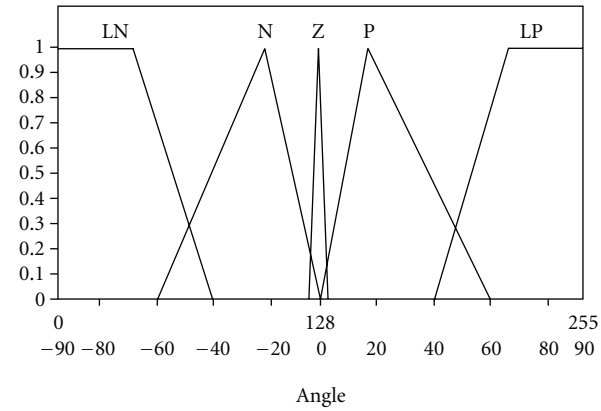
C language. As the operating system we used the kernel of Xilinx standalone.

Experiment 4. The Power PC hard processor of the FPGA Virtex 5 was used. The experimental board "ML507 FPGA" provided us with 16 MB of RAM and 2 KB of cache memory; and the system clock is 100 Mhz. The FLC was implemented in C language. As the operating system we used the kernel of Xilinx standalone.

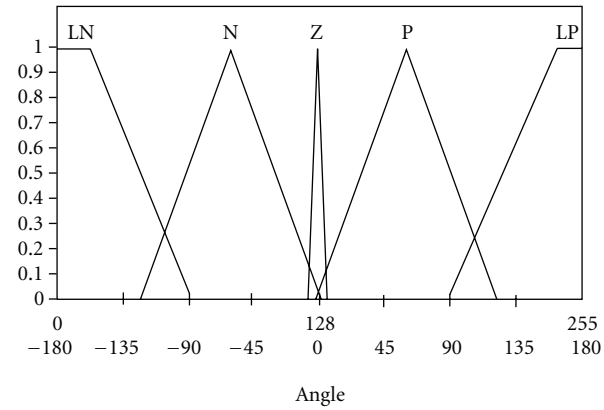
Experiment 5. The idea behind this experiment is to evaluate the FLC implemented in an 8-bit microcontroller system. We used the Atmel AVR development platform with the general-purpose microcontroller Atmel ATmega 16 running at a frequency of 16 Mhz.

Experiment 6. We used the system architecture of Figure 5 implemented in the Actel development board for FPGA model "Embedded M1AFS1500 KIT." This board is based on the FPGA Fusion of the same company; see Figure 11. This implementation uses the ARM cortex soft-processor and incorporates the FLC IP as a coprocessor. The FLC was coded in VHDL. Figure 11 shows a simple test of the FLC, we chose values with known output. The ARM soft-processor sent to the FLC, through the GPIO IP the input values "Error = 69.3 ($0 \times$ EF)", "change of error = −60.5 ($0 \times$ 1F)", and "$k = 0.5$ (0×07)". The output value of −60 (0×51) is read by the ARM and sent to the OLED display (red rectangle).

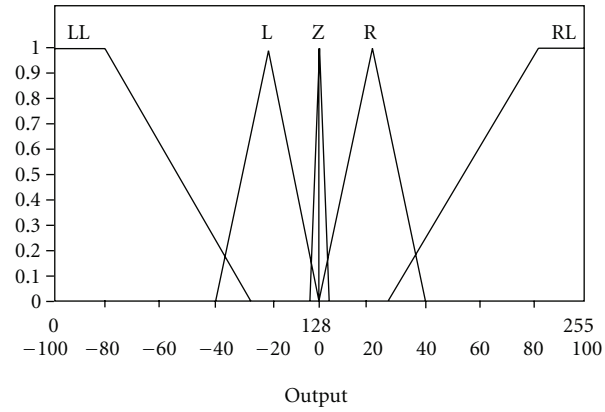
6.1.2. Set 2 of Experiments. Several experiments for different k values using the FPGA Fusion configured as in Figure 5 were achieved. The ARM soft processor handling the FLC IP



(a) Input error



(b) Change of error



(c) Output

Figure 16: The FLC has two inputs: Error and Change of error, and one output. The output provides a value of increment/decrement to be used by the integral part of the PD+I controller. The first scale [0.255] are the real MFs bit values definition in the FLC. The second scale $[-90, 90]$, $[-180, 180]$, and $[-100, 100]$ are the operation values for this application. The scale conversion is achieved by the ARM processor.

was tested. The system response was modified using the STA. The ARM processor modifies the k tuning factor though the GPIO IP.

Results for k values of 0.5, 0.7, 0.8, and 1.0 are shown. Figures 12 and 13 show the system response and errors, for the mentioned tuning factors. Note that $k = 0.8$ provides the

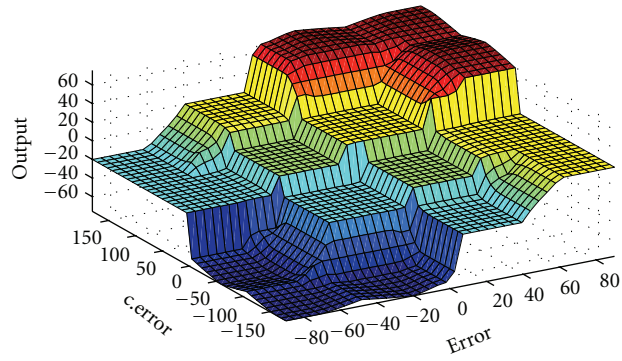


Figure 17: Surface control of the inverted pendulum system.

best system response; whereas $k = 0.5$ the slower response, and $k = 1.0$ the faster response with an overshoot. Figure 14 shows the control surface of the controller for a k value of 0.5. The sampling time for this set of experiments was 700 μs.

6.2. System 2: Inverted Pendulum on a Cart. For system 2 the inverted pendulum on a cart system of Figure 4 was used. We repeated all the experiments of Table 1 for this system; in congruence with the results of this table, we obtained practically the same running times in all the experiments. Next, we show the experimental setup for the FPGA Fusion. The membership functions were tuned using the STA.

The FLC has two inputs, "Error" and "Change of error" and one output labeled as "output". Figure 15 shows the rule matrix for this system, in Figures 16(a), 16(b) and 16(c) are the membership functions for the inputs and output linguistic variables, and Figure 17 shows the surface control of the inverted pendulum on a cart system.

7. Conclusions

The design and implementation of a High-Performance FLC that incorporates an efficient and practical method to tune the controller to the plant was explained. The tunable FLC IP was integrated into a SoC based on the FPGA Fusion jointly with an ARM soft processor and tested using two electromechanical systems.

In the first system, the regulation of the speed of a DC motor was the control objective, being the target of the paper to evaluate the performance of hardware implementations of fuzzy controllers used as IP Cores, two sets of comparative experiments of the system performance were achieved. The aim of the first set was to compare the FLC IP coprocessor against other development platforms, including a Core 2 Duo PC system, three implementations of the FLC programmed in C language running on the Microblaze soft-core processor mounted into the Spartan 3, and Virtex 5 FPGA systems. One C language implementation of the FLC running into the Atmel AVR 8-bit Microcontroller, and one implementation of the FLC coded using VHDL, and mounted as IP into the FPGA Fusion. We achieved an experimental evaluation of time computational complexity of the above systems.

The worst performance system was the FLC implemented using the Atmel AVR 8-bit Microcontroller, with this experiment we got the worst-case bound.

In the average bound, we found the three software prototypes of the FLC programmed in C language running on FPGA platforms, together with the PC development. Differences among FPGA systems are mainly due to systems clock; this is more evident on the Spartan 3 running with half of the frequency clock with respect to other FPGA systems. In the PC, the problem was that the FLC was running on Simulink controlling the plant using serial communication. Due to the implementation characteristic, this was not a real time system.

In the second system, the same set of tests were achieved and the execution times were consistent with the results obtained in the first system, some nonsignificant changes in values of FLC implemented in C occurred. However; for the case of study, the FLC IP Core, always last the same time, this is because the execution time is tight to the system clock, the FLC always last three clock cycles, and the GPIO IP last five clock cycles, therefore the FLC IP last eight clock cycles.

The FLC IP was designed to work in real time, when the FLC finish a whole inference cycle (fuzzification, inference, defuzzification) it enables the IRQ output request that can be sent to the interrupt controller for real time applications, or latched to be read by an input/output port to work in polling mode for applications that does not require real time.

The best-case SoC is the FLC IP combined with the ARM soft-processor, the speed-up with respect to the other development platform is 78, 125 times.

There are other interesting points to remark, for example, the flexibility that FPGA based system provides to implement SoC; this is important because it is possible to design independent low-cost low-power consumption and inexpensive systems that can mix digital and analogical signals. The use of specialized software, the growing availability of resources such as IP allow to simplify the development process of real time systems. The FLC IP soft-core has the advantage that it is not dependable on the bus system, and it provides to the user an easy handling capability without the need of the knowledge of the internal functionality; on the other hand, because it is not encrypted, the content can be read.

Finally, the incorporation of specialized debugging and testing modules into the SoC facilitates these complex and tedious tasks.

At present time, we are developing the IP core a Type-2 FLC, we have simulated stage by stage, for example the type-2 defuzzification stage [25]. The whole Type-2 FLC can be tuned using the STA algorithm for type-2 [26].

Acknowledgments

The authors would like to thank the "Instituto Politécnico Nacional (IPN)", "Comisión de Operación y Fomento de Actividades Académicas (COFAA)", and the Mexican "Consejo Nacional de Ciencia y Tecnología (CONACYT)" for supporting the research activities.

References

[1] M. J. Flynn and W. Luk, *Computer System Design: System-on-Chip*, John Wiley & Sons, Hoboken, NJ, USA, 2011.

[2] L. Idkhajine, E. Monmasson, M. W. Naouar, A. Prata, and K. Bouallaga, "Fully integrated FPGA-based controller for synchronous motor drive," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4006–4017, 2009.

[3] F. Sun, H. Wang, F. Fu, and X. Li, "Survey of FPGA low power design," in *Proceedings of the International Conference on Intelligent Control and Information Processing (ICICIP '10)*, pp. 547–550, August 2010.

[4] D. Saha and S. Sur-Kolay, "SoC: a real platform for IP reuse, IP infringement, and IP protection," *VLSI Design*, vol. 2011, Article ID 731957, 10 pages, 2011.

[5] L. Tian, H. Pan, and D. Li, "Efficient Memory Processors Design of Multiple Applications for Multiprocessors Architecture," in *Software Engineering and Knowledge Engineering: Theory and Practice in Advances in Intelligent and Soft Computing*, Y. Wu, Ed., pp. 693–697, Springer, Berlin, Germany, 2012.

[6] R. E. Precup and H. Hellendoorn, "A survey on industrial applications of fuzzy control," *Computers in Industry*, vol. 62, no. 3, pp. 213–226, 2011.

[7] J. Kackprzyk, "Multistage fuzzy control: a model-based approach to fuzzy control and decision making," *Journal of Multi-Criteria Decision Analysis*, vol. 7, no. 4, pp. 239–240, 1998.

[8] T. J. Ross, *Fuzzy Logic With Engineering Applications*, John Wiley & Sons, Singapore, 3rd edition, 2010.

[9] L. Jim, *Embedded Control Systems in C/C++*, CMP Books, Berkley, Calif, USA, 2004.

[10] V. Thareja, M. Bolic, and V. Groza, "Design of a fuzzy logic coprocessor using handel-C," in *Proceedings of the 2nd IEEE International Workshop on Soft Computing Applications (SOFA '07)*, pp. 83–88, Oradea, Romania, August 2007.

[11] P. Sundararajan, *Performance Computing Using FPGAs*, Xilinx, San Jose, Calif, USA, 2010, http://china.origin.xilinx.com/support/documentation/white_papers/wp375.

[12] R. Sass and A. G. Schmidt, *Embedded Systems Design With Platform FPGAs*, Elsevier, New York, NY, USA, 2010.

[13] S. Anvar, O. Gachelin, P. Kestener, H. Le Provost, and I. Mandjavidze, "FPGA-based system-on-chip designs for real-time applications in particle physics," *IEEE Transactions on Nuclear Science*, vol. 53, no. 3, pp. 682–687, 2006.

[14] O. Montiel, J. Olivas, R. Sepúlveda, and O. Castillo, "Development of an embedded simple tuned fuzzy controller," in *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ '08)*, pp. 555–561, June 2008.

[15] O. Montiel, Y. Maldonado, R. Sepúlveda, and O. Castillo, "Simple tuned fuzzy controller embedded into an FPGA," in *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS '08)*, pp. 1–6, May 2008.

[16] R. Sepúlveda, O. Montiel, O. Castillo, and P. Melin, "Embedding a high speed interval type-2 fuzzy controller for a real plant into an FPGA," *Applied Soft Computing Journal*, vol. 12, no. 3, pp. 988–998, 2012.

[17] Y. Maldonado, O. Castillo, and P. Melin, "Optimization of membership functions for an incremental fuzzy PD control based on genetic algorithms," in *Soft Computing For Intelligent Control and Mobile Robotics*, O. Castillo, J. Kacprzyk, and W. Pedrycz, Eds., vol. 318, pp. 195–211, Springer, Berlin, Germany, 2011.

[18] N. R. Cázarez-Castro, L. T. Aguilar, and O. Castillo, "Fuzzy logic control with genetic membership function parameters optimization for the output regulation of a servomechanism with nonlinear backlash," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4368–4378, 2010.

[19] I. S. Shaw, *Fuzzy Control on Industrial Systems: Theory and Applications*, Kluwer Academic, New York, NY, USA, 2010.

[20] A. Stefano and C. Giaconia, "An FPGA-based adaptive fuzzy coprocessor," in *Computational Intelligence and Bioinspired Systems, Lecture Notes in Computer Science*, C. Sandoval, J. Cabestany, A. Prieto, and F. Sandoval, Eds., vol. 3512, pp. 1–8, Springer, 2005.

[21] Actel, *The Advantages of the 32-Bit Cortex-M1 Processor in Actel FPGAs*, Actel, 2007, http://www.actel.com/documents/CortexM1_Advantages_WP.pdf.

[22] H. A. Ortiz-De-La-Vega, E. Gomez-Ramirez, and J. C. Cortes-Rios, "Simple Tuning Algorithm improvements for fuzzy logic controllers," in *Proceedings of the 6th IEEE World Congress on Computational Intelligence (WCCI '10)*, pp. 1–8, July 2010.

[23] M. A. P. García, I. M. M. Sanchez, O. Montiel, R. Sepúlveda, and O. Castillo, "Simple tuning of a fuzzy pulse width modulation controller for a DC motor application," in *Proceedings of the International Conference on Artificial. Intelligence (ICAI '06)*, vol. 2, pp. 598–604, Las Vegas, Nev, USA, June 2006.

[24] O. Montiel, R. Sepúlveda, P. Melin, O. Castillo, M. Á. Porta, and I. M. Meza, "Performance of a simple tuned fuzzy controller and a PID controller on a DC motor," in *Proceedings of the IEEE Symposium on Foundations of Computational Intelligence (FOCI '07)*, pp. 531–537, April 2007.

[25] R. Sepúlveda, O. Montiel, O. Castillo, and P. Melin, "Modelling and simulation of the defuzzification stage of a type-2 fuzzy controller using VHDL code," *Control and Intelligent Systems*, vol. 39, no. 1, pp. 33–40, 2011.

[26] E. Gómez-Ramírez, P. Melin, and O. Castillo, "Simple tuning of type-2 fuzzy controllers," in *Soft Computing for Intelligent Control and Mobile Robotics, Studies in Computational Intelligence*, O. Castillo, J. Kacprzyk, and W. Pedrycz, Eds., vol. 318, pp. 103–123, Springer, Berlin, Germany, 2011.