HARDWARE SOFTWARE CO-DESIGN OF FUZZY SYSTEMS

M. Alvarez Mesa, F. A. Rivera Vélez and J. E. Aedo Cobo
malvarez@udea.edu.co, rivera@microe.udea.edu.co, joseaedo@udea.edu.co

Grupo de Microelectrónica. Universidad de Antioquia. Colombia

ABSTRACT

Hardware/software co-design is a modern technique for designing complex electronic systems constituted by hardware and software. This article shows a co-design methodology application in fuzzy systems implementation. It starts with a fuzzy system formal description and different hardware/software implementation alternatives are analyzed according to their cost and performance.

1. INTRODUCTION

In last years, fuzzy systems (FSs) have turned into an useful tool to treat and to model complex and non linear systems. Specially in areas like control, image processing, robotics and consumer electronics, FSs have been incorporated in a great number of products and processes [1] [2]. Due to these reasons, there is a great interest in FSs integration in embedded electronics, designed for specific applications in the mentioned areas.

There are several approaches that can be employed to implement a FS. One of them consists in designing an application specific integrated circuit (ASIC) that executes the FS. This implementation can be made using analog, digital or mixed design techniques [3] [4] [5]. Another approach consist in a software description using a high level language like C or C++. Software execution requires the use of a processor, that can be a general purpose processor or an application specific processor (ASIP) with a fuzzy dedicated instruction set [6] [7].

The previous alternatives differ mainly in obtained performance. An aspect that significantly influences performance is related to the way FS characteristic parameters are defined and stored, as well as operators used in fuzzification, inference, rule aggregation and defuzzification processes [3] [7].

For cost sensitive applications that require high performance, an interesting implementation approach consists in using low cost processors together with ASICs, designed to perform the most demanding parts from a computational view of the FS. This approach is related to a methodology called co-design. An inherent obstacle to this methodology is the selection of a proper hardware/software partition and the availability of tools for system co-simulation and validation. CAD software development to apply this methodology has been object of an intense research in last years [8]. An example of these development tools is Polis system [9]. It includes some tools dedicated to co-design. In this research, co-design methodology is used based on Polis for FSs implementation. For this purpose, two FSs was modeled (one of two inputs and one output, and another one of three inputs and one output) and different hardware/software partition alternatives were analyzed. This article has been structured in the following way: in section 2, a description of co-design methodology is done; in section 3, a description of FSs are developed, and their modeling using Esterel language and co-simulation and co-synthesis process are described. In section 4, an analysis of obtained results is presented. Finally, in section 5, conclusions and future researches are presented.

2. HARDWARE/SOFTWARE CO-DESIGN

Most current digital electronic systems are constituted by hardware and software components. Software components consist of programs developed in a high level language running on a programmable processor. Hardware components consist of ASICs. Hardware/software combination tries to satisfy design specifications related with performance, cost and development time [10]. Traditional implementation of this kind of systems is based on the development of software and hardware components in a separate way. This approach restricts the possibility of developing a wide exploration of design space for each application. Also, it makes difficult hardware and software interconnection design, and it allows a mistake possibility in components final integration process; increasing development cost and time [11].

These difficulties have stimulated new CAD tools development for designing such systems. Using these

tools it is possible to develop concurrent specification, simulation and synthesis of hardware and software components.

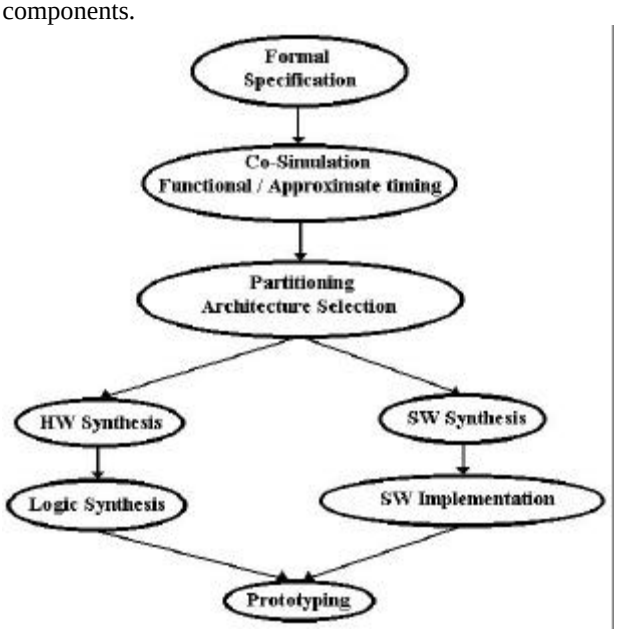


Figure 1. Polis co-design flow

The aim of these new methodologies is to satisfy design specifications at system level, exploiting the existing synergy between hardware and software through its concurrent design [10]. Models and co-design tools differ in the level where re-programming is done.

When it is made in processor application level, co-design consists on software and hardware unified treatment and automatic synthesis, assuming hardware as a co-processor [12].

Within embedded systems field there are a great interest in control oriented systems or reactive systems, that are characterized by receiving events from environment as inputs, and respond generating output events continuously [13]. Design of this kind of systems has real time, cost, performance and power consumption constraints, that can be approached in a consistent way using CAD for co-design. In this article, Polis system was used, whose design cycle is shown in Figure 1. The main steps of this methodology are described next [14].

2.1 Formal specification

Design starts with a system specification in a high-level language. Esterel is the specification language recommended for Polis, a synchronous language for reactive systems with a model of computation based on concurrence with perfect synchrony, in which concurrent processes can execute tasks and interchange information in zero time, at least in a conceptual level [15]. This language allows to specify functionality, independently of hardware/software implementation.

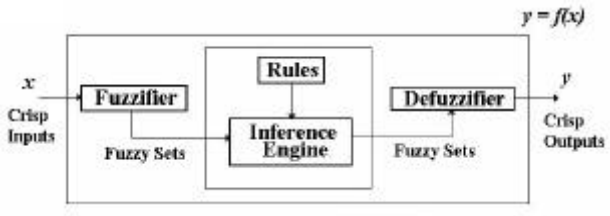


Figure 2. FS internal structure.

2.2 Co-simulation

Polis offers an environment to evaluate alternatives of design through simulation. Models, implemented in hardware or software, are simulated using Ptolemy, which is a simulation and design environment for heterogeneous systems. In Ptolemy, objects are described in different levels of abstraction (domains) using different models of computation. Primitives of each domain, called "stars", can be used in a simulation or synthesis way. The function of each one is described in Esterel. For reactive systems, the most appropriate domain is DE (Discret Event) domain that uses a model of computation events driven [8]. Co-simulation can be made at two levels of abstraction: (1) at a functional level, where time is ignored and it is only important system correct operation; and (2) at an approximate time level, in which execution time is considered through a calculation of execution cycles for hardware and software, according to a master clock.

2.3 Design Partition

In co-simulation process, one can choose in a dynamical way the implementation type of each component and other parameters like processor, clock rate and scheduling algorithm, for satisfying system design constraints. A key aspect of Polis methodology is the possibility of exploring interactively a great variety of partition alternatives, estimating the cost and performance of each one.

2.4 Co-synthesis

Co-synthesis refers to the integrated synthesis of hardware and software partitions. A co-design finite state machine (CFSM) is associated with each module described in Esterel [16]. CFSMs are locally synchronous and interaction among them is globally asynchronous. Each CFSM can be synthesized in hardware or software. For software synthesis, Polis translate each CFSM into a

technology independent high level intermediate representation, well-known as S-GRAPH; then S-GRAPH is translated into standard C code, that can be executed on a processor after compilation.

Polis synthesizes a small operating system, assigning input and output ports for communication between hardware and software, and making tasks scheduling with

R_1: IF X_1 **IS** I_1^1 **AND** X_2 **IS** I_1^2 **AND**... X_M **IS** I_1^M
THEN Y IS C_1
R_2: IF X_1 **IS** I_2^1 **AND** X_2 **IS** I_2^2 **AND** ... X_M **IS** I_2^M
THEN Y **IS** C_2
...
R_N: IF X_1 **IS** I_N^1 **AND** X_2 **IS** I_N^2 **AND** ... X_M **IS** I_N^M
THEN Y **IS** C_N

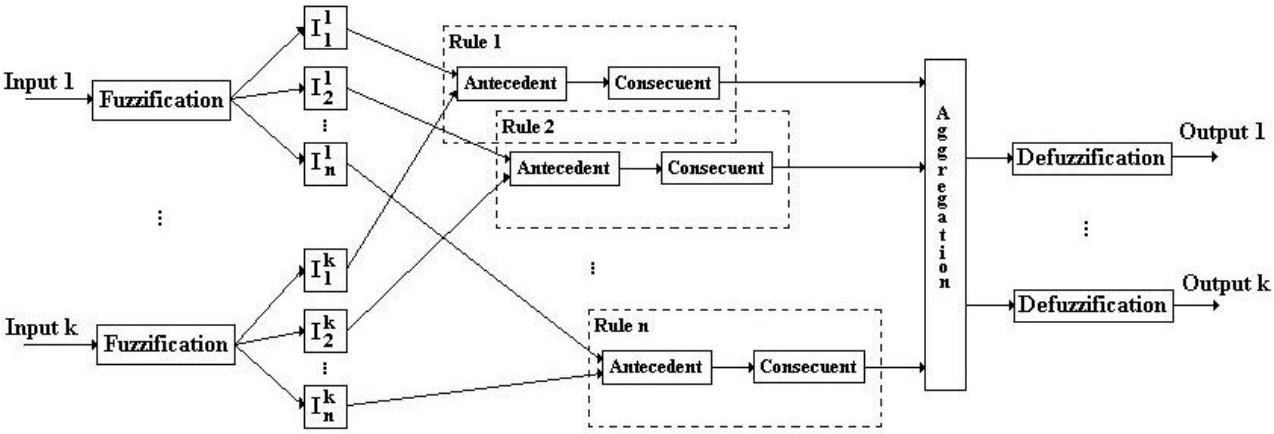


Figure 3. FS execution

an algorithm selected during partition. Synthesis includes code generation in C with scheduler and I/O drivers. Together with synthesis process, POLIS generates for hardware a synchronous digital circuits logical description with BLIF format, which can be used for FPGA programming or VHDL code generation. With synthesis results, it is possible to build a system prototype interconnecting a processor and a programmed FPGA.

3. FUZZY SYSTEMS IMPLEMENTATION USING CO-DESIGN

3.1. Internal structure of the used FSs

Essentially, a FS is constituted by three components: a fuzzificator, a inference machine and a defuzzificator [17]. Figure 2 shows a disposition of these three components. Fuzzification develops a transformation of categorical (real) numbers in fuzzy sets. Inference machine, does the inference process based on a fuzzy rule set, and on an input fuzzy sets generated in fuzzification process. The result of inference is a fuzzy set defined in output space. Finally, defuzzification process generates an output categorical number from the fuzzy set given by inference machine. From the input/output point of view, a FS can be assumed as a transformation (generally not lineal) among input and output real numbers. The typical form of a rule set is the following one:

Equation 3.1. Fuzzy rule set.

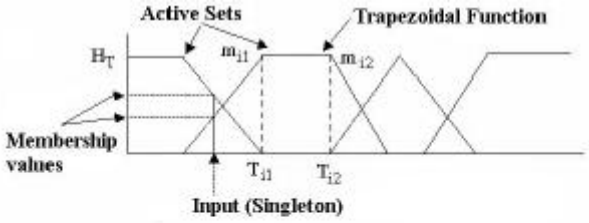
This rule set defines FS dynamics. I_n^m represents the terms defined in input m and in rule n, while C_n represents the terms defined in output of rule n. x_m represents the linguistic variables associated to input n. Rule set illustrated in Equation (3.1) defines a FS of m inputs and one output [18]. Inference machine evaluates the rule set based on fuzzified specific values presented in FS inputs. Basically, the rule set defines a fuzzy relationship between input and output spaces [7].

Figure 3 shows the steps followed in FS execution. In this figure, fuzzification process is initially carried out, according to that, it is established the input singleton membership degree related to each term (fuzzy set I_n^k) defined for that input. Assuming an overlap degree equal to two among membership functions, the maximum number of terms presenting a bigger intersection than zero with a singleton is two. Each fuzzy set group is described with two points and two slopes. Membership value is determined according to the interval where singleton is positioned. This calculation is illustrated in Figure 4.

Once determined this intersection value, rules activation degree is calculated. For this, it is verified if active terms are defined in rules antecedents. If that is right, rule activation value is calculated through the minimum value determination among two intersection

values found in the corresponding terms. This procedure is applied to all the rules.

Finally, the contribution of each rule is added to generate the inferred fuzzy set by the whole set of rules. Inferred set is defuzzified for calculating a categorical output. For the FSs implemented in this work, a singleton in output (Si) is associated to each rule. In this way, output
value is calculated using the gravity center-based defuzzification method, pondered by activation degree (W_i), like it is show in Equation (3.2).



Trapezoidal function membership:

$$\mu_{Ci}(x) = \begin{cases} \max(0, H_T - m_{i1}(T_{i1} - x)) & \text{if } x < T_{i1} \\ \max(0, H_T - m_{i2}(x - T_{i2})) & \text{if } x > T_{i2} \\ H_T & \text{if } T_{i1} \geq x \geq T_{i2} \end{cases}$$

Figure 4. Membership calculation for an input singleton.

$$y = \frac{\sum_{i=1}^{n} W_i S_i}{W_i}$$

Equation 3.2. Defuzzification method.

3.2. Esterel modeling

In the realization of this work, two FSs benchmark were used, whose characteristics are shown in Table 1. These FSs were used to validate the implementation methodology.

Model	Inputs / Terms	Outputs / Terms	Rules
FS1	2 / 3-3	1 / 5	7
FS2	3 / 4-3-6	1 / 5	80

Table 1. FS Benchmarks

In Figure 5, FS1 model is shown inside Ptolemy environment . For modeling these FSs in Esterel, the fuzzification stage was divided in two modules by each input. The first of them (fuzz_control1) examines all defined terms for the corresponding input, and gives to second module (fuzz_calc1) the parameters that describe membership functions (slopes and points). With these data and the sampled input value, the second module defines if intersection exists and its degree. Each fuzzification unit offers information about active terms (maximum 2) and its membership (activation) value.

For rules processing, two modules were created: one of them is called "Rules" and carries out a searching in all defined rules, it establishes if active terms coincide with rule antecedents. If that is true, it calculates the activation degree through minimum operator. This activation degree is multiplied by the output singleton value defined for each

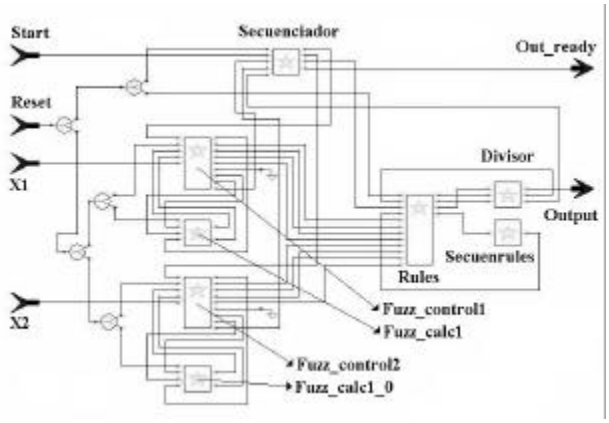


Figure 5. FS1 in Ptolemy.

rule, and then its value is accumulated. Likewise, activation degree is accumulated with each rule contribution. These accumulated values correspond to (3.2) equation numerator and denominator. The other module (divisor) was defined to carry out division among accumulated values.

It has been created a control module (secuenciador) for keeping whole system synchronization, generating the appropriate sequence of events for system operation.

This same procedure was carried out for FS2. It should be noticed that is possible to reuse fuzzification modules previously modeled in Esterel.

3.3. Co-simulation using Ptolemy

Initially, a hardware implementation of the whole FS was made for a functional co-simulation and to verify the description made in Esterel. Figure 6 shows the obtained control surface using the Matlab fuzzy logic toolbox, that serves as a reference to compare the results obtained in FS1 co-simulation, illustrated in Figure 7.

After FSs functional verification, it was carried out a time simulation to estimate software cost, using an

approximate processor model (in this case the MC68HC11E9). The aim of this simulation is to analyze if real time restrictions are satisfied.

As starting point, it is useful to evaluate the whole system implementation completely in software, then one decides if it is necessary to increase the performance of one or several modules, implementing them in hardware.

3.4. Design Partition

POLIS offers tools for analyzing the software components performance in terms of its activation times and response ability to high rate of input events. This analysis shows that membership calculation modules in fuzzification stage, rule processing module, and output categorical value calculation module in defuzzification stage, have the

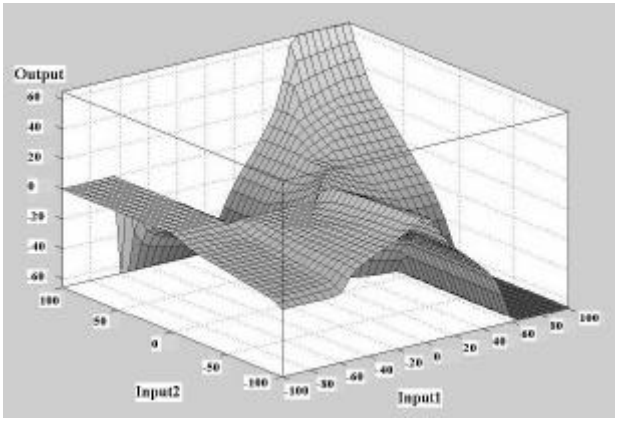


Figure 6. FS1 Control surface obtained with Matlab fuzzy logic toolbox.

biggest computational complexity. When time restrictions don't achieve to be satisfied, these modules are chosen for being implemented in hardware. An additional restriction is that rule processing module cannot be implemented in hardware because it contains C language procedures for rules parameters loading.

According to previous analysis, we decide to implement fuzzification (fuzz_calc) and defuzzification (divisor) modules in hardware. The other modules are implemented in software, and they will be executed on the MC68HC11E9 8 bits microcontroller with an 8 Mhz clock frequency. Round Robin [19] is chosen as a scheduling algorithm for the real time operating system, which carries out a tasks sequential execution without taking into account priorities.

3.5. Co-synthesis

In the software synthesis process, specific code optimizations for reactive systems are carried out. In this level, a software computational cost estimate is made, represented by maximum and minimum execution times and code size. Estimates for FS1 and FS2 are shown in Tables 2 and 3, with all modules implemented in software.

MODULE	MINIMUM TIME (CYCLES)	MAXIMUM TIME (CYCLES)	SIZE (BYTES)
FUZZ_CONTROL1	343	1043	1617
FUZZ_CONTROL2	343	1043	1609
FUZZ_CALC 0/1	546	1556	294
DIVISOR	134	424	183
SECUENRULES	104	146	97
RULES	423	1403	3196
SECUENCIADOR	129	322	618
TOTAL	1749	5159	7467

Table 2: FS1 software implementation cost.

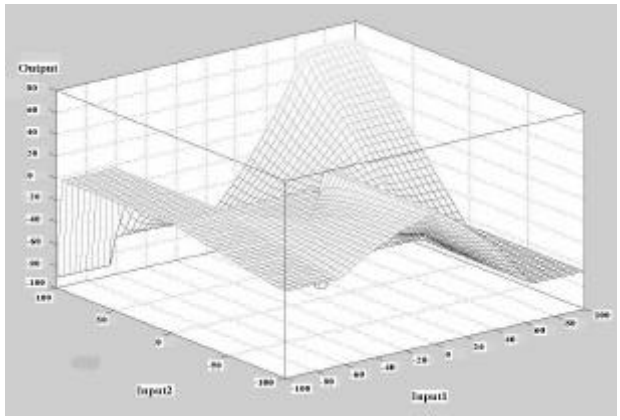


Figure 7. FS1 Control surface obtained via functional co-simulation.

When fuzzification (fuzz_calc) and defuzzification (divisor) modules are implemented in hardware, software execution time and code size are reduced like it is shown in Table 4.

MODULE	MINIMUM TIME (CYCLES)	MAXIMUM TIME (CYCLES)	SIZE (BYTES)
FUZZ_CONTROL_0	343	1043	1617
FUZZ_CONTROL2_0	343	1043	1609
FUZZ_CONTROL3_0	343	1043	1609
FUZZ_CALC 0/1/2	819	2334	3441
SECUENRULES	104	146	97
SECUENCIADOR	154	146	97
RULES	503	1720	10144
DIVISOR	134	424	183
TOTAL	2197	6343	16503

Tabla 3. FS2 software implementation cost.

During the RTOS synthesis, communication between hardware and software tasks is made by interrupts. Interrupt service routines (ISRs) are also automatically generated. RTOS synthesis assigns microcontroller ports for hardware/software interface too. 53 and 69 bits are exchanged between hardware and software in FS1 and FS2, respectively.

IMPLEMENTATION	MINIMUM TIME (CYCLES)	MAXIMUM TIME (CYCLES)	SIZE (BYTES)
SOFTWARE:			
FS1:	1749	5159	7467
FS2:	2197	6697	17348
PARTITION:			
FS1:	1337	3947	6957
FS2:	1790	5495	16018

Tabla 4. Implementation cost: software v.s. partition.

Modules synthesized in hardware are represented in BLIF format. Table 5 shows hardware cost and a reduction through logical simplification. Results of hardware synthesis are expressed in number of input ports (pi), number of output ports (po), nodes, latches and number of literals in sum of products form (lits). These parameters are used in a rapid prototyping environment to decide what type and number of FPGAs are required to implement the selected partition.

	PI	PO	NODES	LATCHES	LITS
NOT OPTIMIZED					
FS1	34	13	5184	164	38137
FS2	32	15	14582	378	105476
OPTIMIZED					
FS1	34	13	4370	164	261163
FS2	32	15	6565	246	39570

Table 5: FSs hardware cost.

4. RESULTS ANALYSIS

From a functional point of view, FSs presents results in agreement with theoretical ones, like it is shown in control surfaces of Figures 6 and 7. Figure 6 was obtained with the Matlab fuzzy logic toolbox, using a floating point data representation. Figure 7 was obtained based upon co-simulation results, using an 8 bits integer representation. In spite of quantization errors, this representation facilitates efficient arithmetic operations in 8 bits microcontrollers and it reduces hardware portions size.

From synthesis point of view, our implementation can be evaluated through both software computational cost and hardware size. As reference, in Table 6 synthesis data obtained with Polis are compared with results presented by FuzzyTech for the FS1, using the same processor (68HC11), an 8 Mhz clock frequency and an 8 bits integer representation.

SYSTEM	MINIMUM TIME (CYCLES)	MAXIMUM TIME (CYCLES)	SIZE ROM (KBYTES)
FUZZY-TECH	1.1	1.3	0.64
POLIS	0.1	0.6	6.9

Table 6. FS1 Implementation performance: Fuzzytech v.s. Polis

Selected hardware/software partition, shown in Table 4, reflects partition effects on total execution time, where computationally heavier modules are taken to hardware. Execution time obtained allows to develop fuzzy controllers for real time systems in which software implementations are not fast enough.

5. CONCLUSIONS

Hardware/software co-design represents a new methodological perspective in complex digital system design. Thanks to an unified and implementation independent representation it is possible a wide exploration of different alternatives of hardware/software implementations, allowing to satisfy design specifications at a system level. In this way co-design allows to reduce development time and to increase products quality and reliability.

Availability of microcontrollers that contain peripherals for control applications (timers, AD converters, etc.) and programmable logical devices as FPGAs makes possible a rapid prototyping of co-designed systems. In FSs case, co-design demonstrates to be an appropriate methodology when it is necessary to obtain efficient implementations for embedded systems with performance, cost and development time constraints, where FS is just a part of total system.

The implementation carried out with Polis demonstrated to be functionally valid and efficient in hardware and software resources use. Nowadays we are working in a development platform using low cost microcontrollers and FPGAs and allowing prototypes implementations for practical applications.

6. ACKNOWLEDGMENTS

This work has been developed with support of *Comité para el Desarrollo de la Investigación (CODI)* of the *Universidad de Antioquia*..

7. REFERENCES

[1] C. V. Altrock, *Fuzzy Logic and neuro-fuzzy applications explained*, Prentice Hall, Englewood Cliffs, USA, 1995.

[2] C. V. Altrock, "Fuzzy logic and neuro-fuzzy technologies in appliances", *Embedded Systems Conference*, http://www.esc.com, USA, 1999.

[3] A. Kandel and G. Langholz, *Fuzzy hardware, architectures and applications*, Kluwer Academic Publishers, 1998.

[4] A. Costa et al., "Hardware solutions for fuzzy control", *Proceedings of the IEEE*, Vol. 83, No.3, Mar.1995.

[5] G. Ascia, V. Catania, G. Filici, Sergio Plazzo et al, "A VLSI fuzzy expert system for real-time traffic control in ATM networks", *IEEE Transaction on fuzzy systems*, Vol. 5, No.1, Feb. 1997.

[6] V. Salapura and M. Gschwind. "Hardware/software co-design of a fuzzy RISC processor". *In Design Automation and test in Europe Conference*, Paris, France, Feb. 1998.

[7] J. E. Aedo, "Técnicas de implementación en hardware de sistemas difusos", *Seminario internacional sobre procesos avanzados de manufactura*, Medellín, Nov. 2000.

[8] S. Edwards, L. Lavagno, E. A. Lee, A. Sangiovanni-Vicentelli, "Design of embedded systems: formal models, validation and synthesis", *Proceedings of the IEEE* , Vol. 85, No 3, Mar. 1997.

[9]. UC Berkeley, "POLIS: A framework for hardware/software co-design of embedded systems", http://www-cad.eecs.berkeley.edu/~polis.

[10]. G. De Michelli and R. K. Gupta, "Hardware/software co-design", *Proceedings of the IEEE*, Vol. 85, No 3, Mar. 1997.

[11] S. Kumar, *The codesign of embedded systems: a unified hardware software representation*, Kluwer Academic Publishers, Norwell, USA, 1996.

[12] M. Chiodo, P. Giusto, A. Jurecska et al, "Hardware/software codesign of embedded systems", *IEEE Micro*, Ago. 1994.

[13]. G. Berry, P. Couronne, and G. Gonthier, "The asynchronous approach to reactive and real time systems", *Proceedings of the IEEE* , Vol. 79, Sep. 1991.

[14] F. Balarin, M. Chiodo, D. Engels et al. *POLIS, a design environment for control-dominated embedded systems, version 0.4, User's manual*, UC Berkeley, Nov. 1999.

[15] G. Berry, The foundations of ESTEREL, In G. Plotkin et al. Ed., *Proof, language and interaction: essays in honour of Robert Milner*, MIT Press, USA, 2000.

[16] M. Chiodo, P. Giusto, H. Hsieh et al. "A formal methodology specification model for hardware software codesign", *Technical report UCB/ERL M93/48*, UC Berkeley, Jun. 1993.

[17] L. X. Wang, *Adaptive Fuzzy Systems and Control*, Prentice Hall, Englewood Cliffs, USA 1994.

[18] J. M. Mendel, "Fuzzy logic systems for engineering: a tutorial", *Proceedings of the IEEE* , Vol. 83, No 3, Mar. 1995.

[19] J. Labrosse, *MicroC/OS-II. The real time kernel*, R&D Books, Lawrence, USA, 1999.