Mathware & Soft Computing 6 (1999) 319-329

Fast Prototyping of Computing-oriented Fuzzy Logic Systems

J.I. Martínez Torre*, J.-P. Deschamps*, M. Fernández Centeno**
* Escuela Superior de Ciencias Experimentales y Tecnología
Univ. Rey Juan Carlos de Madrid, 28939 Móstoles, Spain
** Dpto. Arquitectura de Computadores y Automática
Univ. Complutense de Madrid, 28040 Madrid, Spain

Abstract

In this paper we propose to explore different dedicated hardware solutions in terms of area-speed trade-offs for computing-oriented fuzzy logic systems in the algorithmic abstraction level that fulfil the application requirements applying high level synthesis techniques. The main benefits of this proposal are the capability of selecting the best implementation for any application not being tied down to a predefined architecture, the use of the required number of hardware units and the reduction of the design cycle time. The results obtained in the control of a maximum power point of a photovoltaic plant are presented as an example.
Keywords: Fast prototyping, dedicated hardware, high level synthesis, hardware reuse, scheduling, design cycle.

1 Introduction and Methodology

There are an increasing number of applications (consumer, industrial, scientific, etc.) implemented thanks to fuzzy logic techniques [1] due to the difficulty or even impossibility of obtaining suitable mathematical models of the application's functionality.

There are several ways of implementing fuzzy logic systems from those with greater programmability (software) to those with greater specialization (dedicated hardware). Although most of the applications are software solutions running on general-purpose processors or controllers, the number of applications with very strict requirements, mainly in terms of speed or area consumption, is growing every day. Those applications need to be implemented not only in hardware, but even in dedicated hardware [2,3] on ASICs and FPGAs. In this last field, most of the proposals in the literature tend to implement a fuzzy logic system into a predefined architecture where the development work amounts to the set up of the parameters of the control unit and the data path structure depending on the application requirements. Usually, these implementations involve more hardware

Figure 1: proposed methodology.

The methodology aims at specifying the fuzzy logic system in an abstact text specification style and automatically generating a fuzzy logic system behavioural description in the algorithmic level (system level); then, appying High Level Synthesis techniques in order to evaluate different implementations and get the one that better suits the system requirements (algorithmic level), and finally synthezise the circuit (low level).

2 FLS Specification

In photovoltaic plants supplied with batteries usually used for terrestrial applications (basically, solar panels and dc-dc converters), the maximum performance is obtained if the maximum power point is always the set up point of the photovoltaic plant, i.e., a maximum power point tracker system is compulsory. This FLS is implemented with two inputs (E and CE), generates one output (dD). Figure 2 shows

the membeship functions (MBF) and rule base.

The specification of an FLS requires setting up a great number of parameters. Some of them are fixed during the verification of the FLS functionality (for instance, the inference method) whilst other parameters depend on the implementation style or the environment where the FLS is going to be working (for instance, memory-oriented or computing-oriented defuzzyfication style). The selection of these parameters guides and greatly influences the next steps of the design flow. For instance, the computing-oriented defuzzyfication style entails the design of dedicated hardware modules in order to generate the degree of membership values in terms of (en función) the number and complexity of the membership functions. The algorithm used to generate the degree of membership values, the characteristic points selected, etc. provide new details and requirements that influence the final results.

dD	CE				
	NB	*NS*	*ZO*	*PS*	*PB*
NB	ZO	ZO	NB	NB	NB
NS	NS	ZO	NS	NS	NS
ZO	NB	ZO	ZO	ZO	PS
PS	ZO	PS	PS	ZO	PS
PB	PB	PB	PB	ZO	ZO

E

Figure 2: Rule base

3 Generation of a FLS description at the algorithmic abstraction level

The FLS specification is processed in order to generate a FLS description in the algoritmic abstraction level, thanks to a set of of three different software programs (Figure 3).

The first one analyses the lexical and syntactical correctness of the FLS specification detecting input, output or membership function name duplications, incoherent antecedents, wrong rules, etc.

The second one optimises the number of lattice operations involved in the inference method, because a direct translation of the lattice equations tends to use more lattice operators than the optimal number.

There are some propositions of the lattice calculus that can be used to reduce the length of the original expression as well as preventing the redundancy of intermediate expressions in the FLS computing algorithm.

Figure 3: System level tools

The reduction in operators (presumably hardware modules, nets, delays and power consumption) can be remarkable, as is shown in Figure 4 from the example in (1).

$$w_1 = x_1 \wedge x_2 \wedge x_5 \vee x_1 \wedge x_2 \wedge x_4 \vee x_3 \wedge x_5 \vee x_3 \wedge x_4$$
$$\text{equivalent to } w_2 = (x_1 \wedge x_2 \vee x_3) \wedge (x_4 \wedge x_5) \quad (1)$$

The basics of this optimizition algorithm - a heuristic algorithm based on the existence of two canonical expressions for each lattice function - were reported in an earlier publication [6].

Finally, the third module performs a syntax-guided translation of the FLS specification, generating a behavioural description in the algorithmic abstraction level in terms of the operations involved in the FLS algorithm.

All the modules were implemented in C, using LEX and YACC tools running on SunOS workstations.

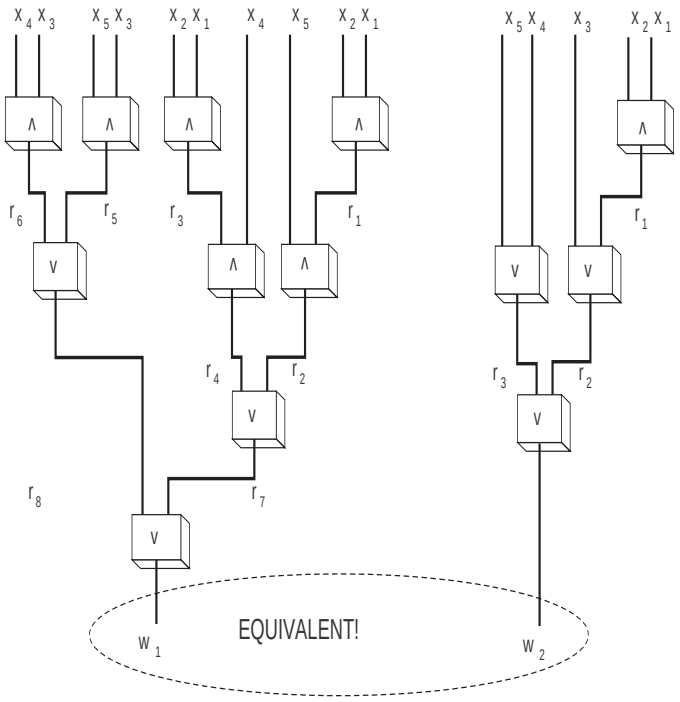

Figure 4: Lattice operations equivalence.

4 Synthesizing from the algorithmic abstraction level: applying HLS

An example of the main parts of the generated description for the photovoltaic plant is shown in Figure 5 in the Fidias [7] input language.

This code represents the steps of the computing-oriented FLS algorithm. First, the input values (port) are loaded into internal variables. Second, the MBF critical points are retrieved. Third, the degrees of MBF are computed on-line and on-chip. Fourth, the fuzzy conclusions are inferred. Fifth, the centres of gravity of the output MBFs are computed. Sixth, the arithmetic equations for the outputs are computed; and finally, the computed output values are written into the output ports.

```
SPECIFICATION SPMP

INPUTS
E NB NS ZO PS PB UOFD -1 +1 PREC 8 MBFV 0 1 PREC 4
PATH Epath
CE NB NS ZO PS PB UOFD -1 +1 PREC 8 MBFV 0 1 PREC 4
PATH CEpath

OUTPUTS
dD NB NS ZO PS PB UOFD -1 +1 PREC 8 MBFV 0 1 PREC 4
PATH CEpath

RULES
IF E is NB and CE is NB THEN dD is ZO
IF E is NB and CE is NS THEN dD is ZO
IF E is NB and CE is ZO THEN dD is NB
...
IF E is PB and CE is PB THEN dD is ZO

FUZZIFICATION Singleton
INFERENCE Min-Max
DEFUZZIFICATION WACoG
```

Figure 5: Problem specification.

These Fidias HLS tools were designed by our working group and were selected because of the availability of its source code and the non-existence of other commercial HLS tools. Nowadays, HLS tools like Behavioral Compiler from Synopsys [8] let us take advantage of their full set of CAD tools in an integrated environment, starting from a behavioural description written in VHDL and decreasing the design cycle time. Although the HLS techniques have usually been tested on behavioural descriptions of DSP applications (filters), the properties of the FLS algorithms (straightforward code, high degree of parallelism, no conditional branches, etc.) make them very appropriate for applying HLS techniques to them.The HLS techniques perform a synthesis step from a behavioural description domain at an algorithmic abstraction level to a structural description domain at a register-transfer (RT) abstraction level, made up of a data path unit and a control unit (ALUs, registers, multiplexors, memory blocks, etc.). As can be seen in Figure 6, the design solution space is evaluated thanks to the HLS tools in order to find the architecture that best suits the requirements.

The first task is to schedule the FLS algorithm, not only according to the user requirements, but also to the available functional units: different hardware libraries with several functions, areas, timings, etc. The scheduler assigns every operation to a hardware operator type in a specific clock cycle.

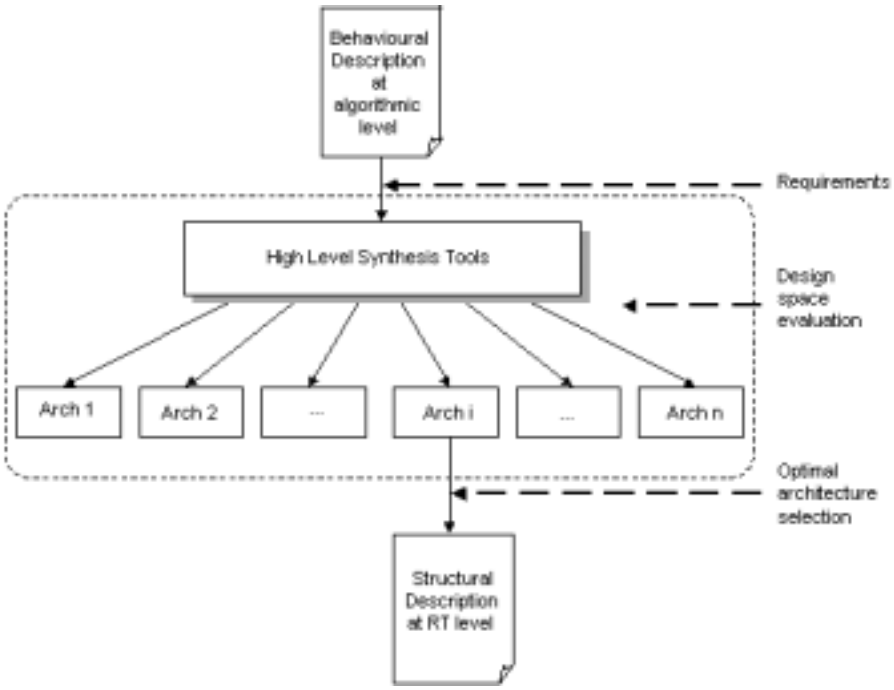

Figure 6: Evaluating the best implementation

Different implementations that vary from those with the maximum degree of parallelism (large area consumed and high throughput) to those with purely sequential computations carried out by one single ALU (small area consumed and low throughput) can be selected. An implementation that uses a great number of operators in parallel for computing the results as soon as possible is shown in Figure 7.

Once the best architecture has been selected, the second task performs the allocation of each operator to a physical hardware unit, each internal variable to a register, and each path into nets and multiplexers, following the data path structure included in Figure 8.

The final task amounts to the design of the control unit that rules the data path operation. A memory and resource allocator accomplishes this task, allowing the data to flow from ports to registers, from registers to operational hardware units, from operational hardware units to registers or from registers to ports. The signals that control all the RT transfers (register operation, multi-operational hardware (+/- or +/*, etc.) and multiplexer) are generated from a wired or microprogrammed control unit, also included in Figure 7. This task concludes generating a description of the selected FLS architeture in the structural domain at the algorithmic abstraction level, ready to follow the usual design cycle targetting the architecture on application specific integrated circuits or field programmable gate arrays.

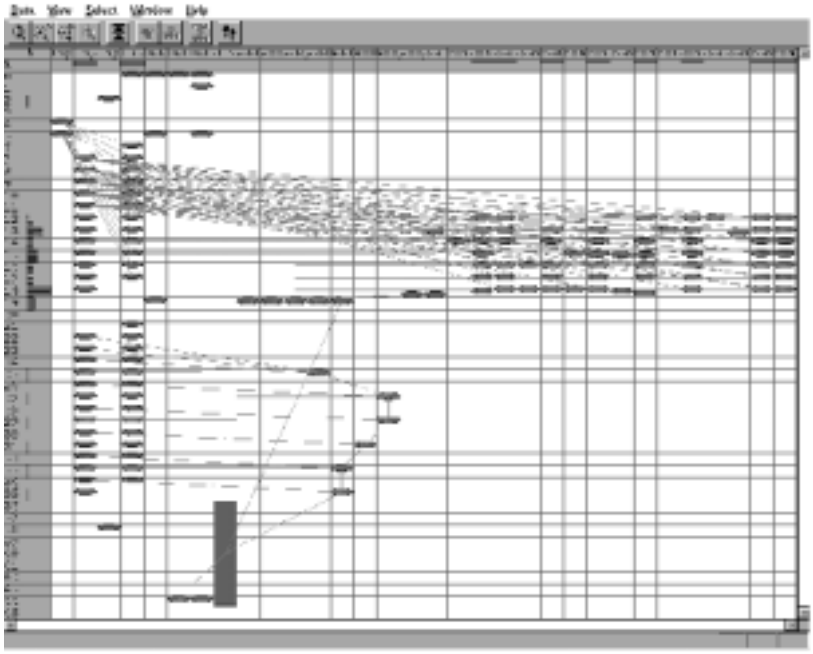

Figure 7: Evaluating the best implementation

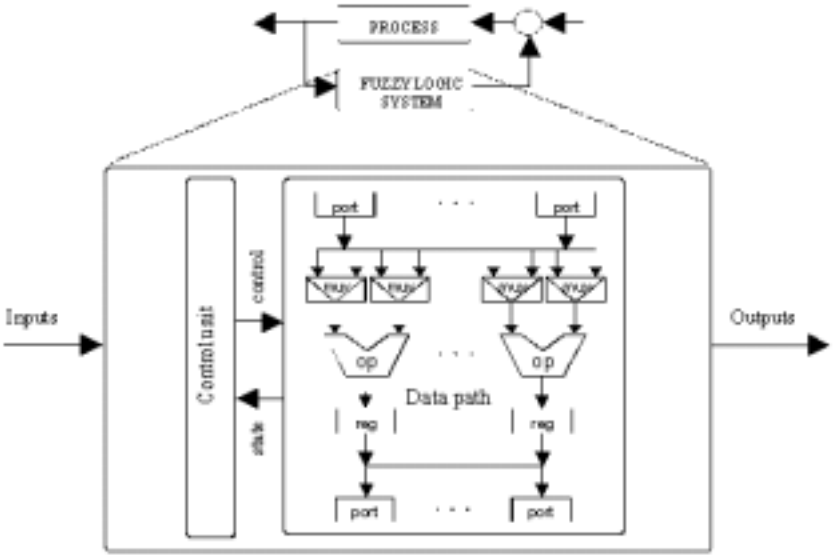

Figure 8: Global view of the solution

5 Evaluating computing-oriented FLS implementations

The major bottleneck in memory-oriented FLS implementations is the number, type and capacity of the memory blocks while on the other hand for computing-oriented FLS implementations it is the complexity of the MBF reconstruction algorithm, and, of course the complexity of the hardware units that perform that task.

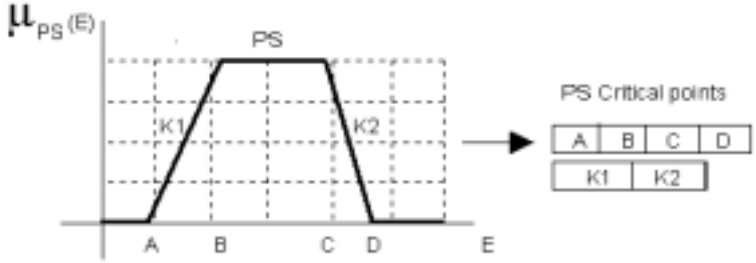

Figure 9: Trapezoidal MBF critical points

In this example, the length of the three variables E, CE (inputs) and dD (output) are 8 bits in C2, while the membership function values are computed in 5 bits. The parameters specified for the FLS were singleton fuzzyfication method in a computing-oriented style, Mandani inference method, and weighted average of centres of gravity (WxGofG) defuzzyfication method in a computing-oriented style, performing the following computations (2).

$$\begin{aligned}
&\text{if i_mbf Then}\\
&\quad \text{If } x \le a_i \text{ or } x \ge d_i \text{ Then } A_i = 0;\\
&\quad \text{else if } a_i \le x \le b_i \text{ Then } A_i = (x - a_i) \cdot k_{i1};\\
&\qquad \text{else if } c_i \le x \le d_i \text{ Then } A_i = (d_i - x) \cdot k_{i2};\\
&\qquad\quad \text{else } A_i = 2^m - 1;;\\
&\text{else if o_mbf Then } C_0 = w_0 \cdot B_0;;
\end{aligned} \tag{2}$$

The shapes of the input membership functions (i_mbf) are stored on-chip as the co-ordinates of four vertexes and two slopes (Figure 9). The input value sampled e' is compared to the appropriate a, b, c and d values and the degree of membership value is computed as 0, 1, (e' - a) $\cdot$ K_1 or (d - e') $\cdot$ K_2.

The simplicity of the output membership functions (o_mbf), defined as isosceles triangles, reduces the computing complexity. The defuzzyfication value is performed as w $\cdot$ B, where B stands for their centre of gravity and is also stored on-chip.

The size of the memory required to store all the input membership functions (fuzzyfication method) and the output centres of gravity (defuzzyfication method) is significantly smaller than the memory size in a memory-oriented FLS implementation. Assuming that all the off-line and off-chip precalculated input and output values are stored in a single memory block, the memory size should be at least 2^8 x 5 + 2^8 x 5 + 2^5 x 8 x 2 bits, instead of 8x6x5x2+5 x 8 bits for the computing-oriented

implementation.

The computing-oriented FLS implementations have several disadvantages and advantages.

The complexity of the algorithms for reconstructing the membership function shapes increases the system response. Also, the on-line adaptability when MBFs change is smaller than the adaptability in memory-oriented designs.

That complexity means more hardware units in the data path in order to compute multiplication, division, etc., but the total area, with less memory resources involved, can be smaller than in the other approach. Also, the FLS implementation can work in stand-alone mode because it is self-contained.

Figure 10 show a set of computing-oriented FLS implementations for the example, ranging from 20 ns to 500 ns clock speeds, using ES2 Ecpd07 standard cells as the target technology.

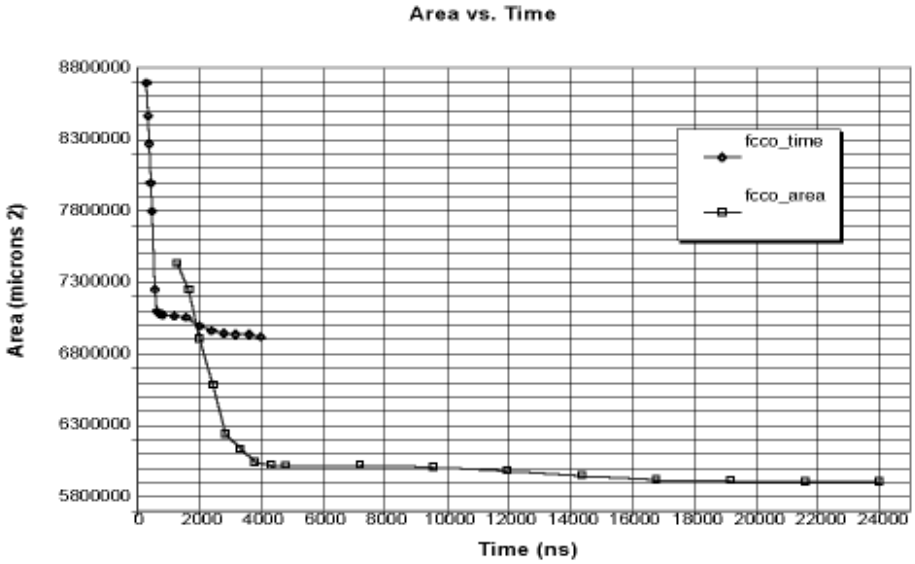

Figure 10: area-time trade-off curves for the computing-oriented FLS implementations

The curve labelled fcco_time represents a set of implementations aiming to produce the results as soon as possible no matter how many hardware units are being used or how much area is being consumed. The curve labelled fcco_area represents a set of implementations aiming to produce the results minimizing the core area no matter how long it takes to produce the results, if the time requirements are not violated. Obviously, there are many more possible implementations in between those curves. For both, the points mean the best implementations for the specific clock speed.

The solution that best meets the requirements can be selected and designed, finishing the design cycle with the layout of the application specific integrated circuit or the bit pattern for the field programmable gate array configuration.

Conclusions

Most of the hardware implementations of fuzzy logic systems usually map the operators of the computing algorithm in a fixed architecture, which has been designed to take advantage of some architectural restrictions.

By using HLS techniques, we are not tied down to using a fixed implementation, but rather we can explore different branches of the design space, as was shown for the computing-oriented FLS implementations, that can be mapped into many different implementations selected through the specification requirements and HLS tools.

References

[1] Thomas D.E., Armstrong-hélovry B, Fuzzy logic lontrol - a taxonomy of demonstrated benefits, Proceedings of the IEEE, vol. 83, n° 3, pp. 407 - 421, 1995.

[2] Ungering A. P., Goser K., Architecture of a PDM VLSI fuzzy logic controller with pipeline and optimized chip area, Second IEEE international conference on fuzzy systems, pp. 447 - 452, 1993.

[3] Jimenez C. J., Sánchez Solano S., Barriga A., Hardware implementation of a General Purpose Fuzzy Controller, Proc. IFSA World Congress, pp. 185-188, Sao Paulo, Brasil, 1995.

[4] Martínez Torre J.I., Deschamps, J-P, Design flow and tools for dedicated fuzzy hardware, Proceedings of IIZUKA'96, pp 255-258, 1996.

[5] Gomariz S., Guerrero J.A., Guinjoan F., Seguimiento del punto de máxima potencia de un sistema fotovoltaico mediante control difuso, Actas de ESTYLF'97, pp 251-256, 1997.

[6] Deschamps, J-P, Martínez Torre J.I., Optimización de operadores reticulares en un entorno de síntesis automática de controladores borrosos, Proceedings of ESTYLF'96, pp 107-112, 1996.

[7] Septien, J., Mozos, D., Tirado, F., Hermida, R., Fernández, M., Mecha, H., FIDIAS: An Integral Approach to High Level Synthesis, IEE Circuits, Devices and Systems, vol. 192, n°4, pp. 227-235, 1995.

[8] Knapp D.W., Behavioral Synthesis: Digital System Design using the Synopsys Behavioral Compiler, Prentice-Hall, 256 pp., 1996.