# An Automated Design Flow from Linguistic Models to Piecewise Polynomial Digital Circuits

Iluminada Baturone
Dpto. Electrónica y Electromagnetismo – IMSE
Univ. of Seville – CNM (CSIC)
Seville, Spain
lumi@imse-cnm.csic.es

Santiago Sánchez-Solano
IMSE
CNM (CSIC)
Seville, Spain
santiago@imse-cnm.csic.es

Andrés A. Gersnoviez, and María Brox
Dpto. Arq. Comput., Electrón. y Tecn. Electrónica
Univ. of Cordoba
Cordoba, Spain
andresgm@uco.es mbrox@uco.es

***Abstract*—This paper describes how the different CAD tools of the environment Xfuzzy 3, developed in Microelectronics Institute of Seville and University of Seville, allow to translate expressive linguistic models into mathematical ones, in particular, into a combination of piecewise polynomial systems that can be implemented efficiently in hardware. The new synthesis tool of Xfuzzy 3 automates communication with Xilinx System Generator in Matlab, thus facilitating implementation of the linguistic model into an FPGA from Xilinx. This is illustrated with the design of a navigation controller for an autonomous robot.**

## I. INTRODUCTION

Model-based approaches are currently gaining popularity to tackle the growing complexity of embedded systems development since they allow working with a high degree of abstraction, enhance understanding and reduce the time to market. Particularly linguistic models, that is, descriptions of systems obtained from heuristic knowledge of human experts expressed linguistically, facilitate understanding and rapid development. In order to translate linguistic models into mathematical ones, which can be implemented in hardware and/or software, fuzzy logic-based systems have been employed widely in the recent years [1]. However, while expressive linguistic models have been employed in many software applications, the models implemented in hardware and embedded software only contain a single (plain) rule base with simple antecedents and consequents, thus reducing the applicability of hardware approaches. This is the case of many design environments for fuzzy systems, such as FIDE, rFLASH, and FuzzyTECH, when generating embedded software for specific processors, and the case of many fuzzy digital circuits.

This paper describes how the different CAD tools of the environment Xfuzzy 3, developed in Microelectronics Institute of Seville and University of Seville [2], ease to fill the gap between the description of expressive linguistic models and its efficient hardware implementation. The idea is to transform successively an initial linguistic model into a combination of plain fuzzy rule bases implemented efficiently by piecewise polynomial digital circuits.

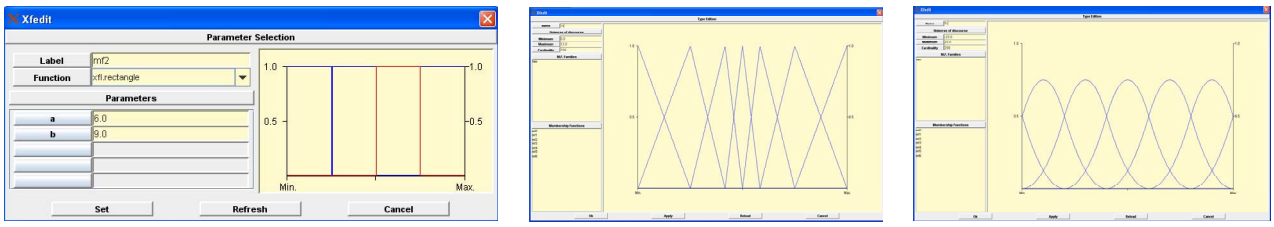
The paper is organized as follows. Section II summaries how plain fuzzy rule bases are equivalent to piecewise polynomial systems (in particular to PWL ones) when imposing certain constraints. Section III describes briefly the different CAD tools of Xfuzzy 3 that facilitate transforming an expressive linguistic model into the combination of piecewise polynomial modules, particularly the new hardware synthesis tool, *xfsg*, which automates the communication between Xfuzzy 3 and the Xilinx System Generator (SysGen) Simulink toolbox of Matlab. Section IV illustrates this automated design flow with the design of a navigation controller of an autonomous robot. Finally, Section V summarizes conclusions.

## II. RULE BASES AND PIECEWISE POLYNOMIAL SYSTEMS

A plain fuzzy rule base is a set of IF-THEN rules whose antecedent parts contain fuzzy evaluations of the input variables. Fuzzy sets are represented by membership functions that define a partition of the input universes of discourse. The sets are fuzzy because their membership functions overlap among them and take values between 0 and 1 (null to full membership). Hence, given an input data, there is a piece of rules with an activation degree greater than zero that is responsible of the output value. Depending on the antecedent and consequent membership functions and the fuzzy operators employed, a plain rule base can perform as a piecewise polynomial system. This performance is a very good trade-off between versatility of the rule base (it features universal approximation capability) and efficiency of its hardware implementation.

Let us consider B-spline families of order zero, one and two to represent the input membership functions. Fig. 1 shows how these functions are normalized, that is, the sum of the membership degrees of any input to them is always the unity. Let us consider that antecedent parts only involve conjunctions that are represented by two operators: the extension of the meet operator [3] and the product. Table I and

This work has been partially supported by European Community under the MOBY-DIC Project FP7-IST-248858 (www.mobydic-project.eu), by Ministerio de Ciencia y Tecnología under the Project TEC2008-04920 and by Junta de Andalucía under the Project P08-TIC-03674.




(a) (b) (c)

Figure 1. B-spline families of order (a) zero, (b) one and (c) two, as defined with the tool *xfedit* in Xfuzzy 3.

II summarize the equivalences between piecewise polynomial approximators and rule bases that employ, respectively, a zero-order Takagi-Sugeno inference method (the rules' consequents are singleton or non fuzzy values) and a first-order inference method (the rules' consequents are linear functions in the inputs). B-splines of degree zero are not a fuzzy solution because there is no overlapping, and B-splines of degree two have low linguistic meaning since total membership is not possible. Hence, B-splines of order one are the most adequate for linguistic models. Using them, piecewise linear, multilinear, quadratic, and multiquadratic systems can be generated [3]-[4]. The plain rule bases considered herein (depicted with bold fonts in Table I and II) use the product as antecedent connective.

Several authors have reported efficient digital architectures to implement such rule bases. In particular, the one considered herein is the active-rule driven architecture reported in [5]. The constituent blocks of this architecture are membership function circuits (MFCs), which can implement B-spline families of order one with uniform or non uniform (Fig. 1b) distribution; rule selection block, which selects the rules activated by the inputs; antecedent connective, which can be selected as the product; rule memory, which stores 1 value per rule in the case of zero-order Takagi-Sugeno inference or n+1 values (being n the number of inputs) in the case of first-order Takagi-Sugeno system; defuzzifier block, which in the case of B-spline families and Takagi-Sugeno systems, can be selected as a weighted sum (no divider is required); and a control block, which generates the control and temporization signals to process sequentially only the $2^n$ active rules.

TABLE I. ZERO-ORDER TAKAGI-SUGENO RULE BASES

| **Antecedent connective** | **Order of B-spline family in antecedents** | | |
|---|---|---|---|
| | **0** | **1** | **2** |
| Extension of meet | Piecewise constant | Piecewise linear (PWL) | Piecewise quadratic |
| Product | Piecewise constant | **Piecewise multilinear** | Piecewise multiquadratic |

TABLE II. FIRST-ORDER TAKAGI-SUGENO RULE BASES

| **Antecedent connective** | **Order of B-spline family in antecedents** | |
|---|---|---|
| | **0** | **1** |
| Extension of meet | Piecewise linear (PWL) | Piecewise quadratic |
| Product | Piecewise linear (PWL) | **Piecewise multiquadratic** |

## III. XFUZZY 3 AND EXPRESSIVE LINGUISTIC MODELS

Plain piecewise polynomial rule bases are adequate to describe a simple linguistic model with 1, 2 or even 3 inputs and with a few membership functions per input, due to the curse of dimensionality. However, complex linguistic models can involve many inputs and membership functions. The first step to facilitate hardware implementation of complex linguistic models should be to use hierarchical systems as much as possible, which combine modules of 1 and 2 inputs, preferably. An advantage of linguistic models is that hierarchy is usually present in the model from the beginning or can be pursued in subsequent refinements. The design environment Xfuzzy 3 helps in this process because it allows describing hierarchical systems consisting of several modules that can interchange fuzzy or crisp information among them. Each module can be a set of fuzzy IF-THEN rules or a non fuzzy (crisp) module described by any mathematical description connecting its inputs and outputs. The CAD tool *xfedit* in Xfuzzy 3 facilitates describing this kind of systems. Fig. 2, for example, shows the main window of the tool *xfedit* when describing a system with three modules (one of them crisp).

The fuzzy modules included in the hierarchical system are not usually piecewise polynomial rule bases if they are obtained from IF-THEN rules expressed in natural language. Such rules in natural language do not always follow a Takagi-Sugeno inference scheme with antecedents represented by B-splines of degree one and only connected by product. Xfuzzy 3 employs a formal specification language, named XFL3, that facilitates translating IF-THEN rules expressed in natural language because of its expressiveness. Not only input variables can be evaluated as fuzzy (for example, 'power is *medium*') but also the fuzzy concepts and the propositions itself can be evaluated linguistically by using linguistic hedges (for example, 'power is *greater than medium* and *more or less* speed is *high* or area is *low*'). The last example in

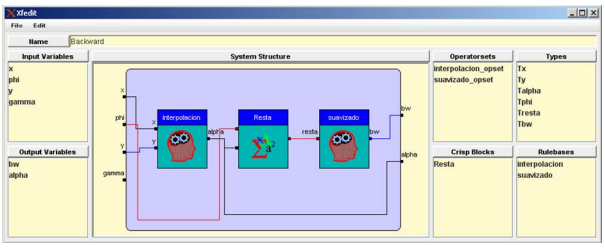



Figure 2. Main window of the tool *xfedit*.

XFL3 would be:

power > medium & ~ (speed == high | area == low)

Another feature of XFL3 expressiveness is that confidence weights can be assigned to the rules. The CAD tool *xfedit* in Xfuzzy 3 contains many graphical user interfaces so as to allow describing complex rules without a deep knowledge of XFL3. The user of Xfuzzy 3 can even define new operators (conjunctive, disjunctive, implication, aggregation, etc.) apart from those already defined in the environment so as to better translate the linguistic meaning of the rules. The CAD tool *xfpkg* in Xfuzzy 3 has a graphical user interface to facilitate the introduction of new operators and crisp modules.

Once the fuzzy modules have an initial mathematical description, the following step is to transform them into piecewise polynomial modules. Such transformation is always possible because piecewise polynomial modules are universal approximators. Several CAD tools of Xfuzzy 3 are very helpful in this process. One of them is the tool *xfplot* that allows representing graphically the output of the linguistic module versus 1 or 2 of its inputs and save the input-output numerical data into a file. These numerical data provided by the linguistic knowledge (and any other numerical data provided by another kind of knowledge) is employed by the tool *xfdm* in Xfuzzy 3 to extract the initial structure of the piecewise polynomial module. This tool contains several grid-based algorithms so as to identify the number and initial distribution of B-splines per input. Tuning of this initial structure to minimize approximation error to numerical data is done with the tool *xfsl*, which includes several supervised learning algorithms [6]. Simplification of the tuned membership functions and rules of each module is done with the tool *xfsp*, which includes pruning, similarity-based, clustering-based and tabulation simplification algorithms [7]. A final tuning is again performed with *xfsl*.

The following step after obtaining a hierarchical system with piecewise polynomial modules is to verify its behavior. Xfuzzy 3 contains several CAD tools to help in this process. One of them is *xfsim*, which uses a model of the context where the system is involved so as to simulate the system dynamically. Another tool is *xfmt*, which allows monitoring each of the constituent rule bases to understand how the output values are inferred from the input ones. The tool *xfplot* is the other verification tool that can be again used to compare the input-output behavior (static behavior) of the transformed and the initial systems.

Static and dynamic verifications in Xfuzzy 3 do not consider hardware implementation aspects such as the influence of parameter word sizes (the static behavior analyzed in Xfuzzy does not consider quantization) and technology details so as to evaluate if the design meet power, area, and speed requirements (the dynamic behavior simulated in Xfuzzy is at high level). Since these details depend on the target platform, our approach has been to exploit the design environments of the hardware platforms and develop a synthesis tool in Xfuzzy whose objective is to automate communication between both environments. In this sense, the new synthesis tool developed for Xfuzzy, named *xfsg*, acts as an interface between Xfuzzy and Xilinx System Generator

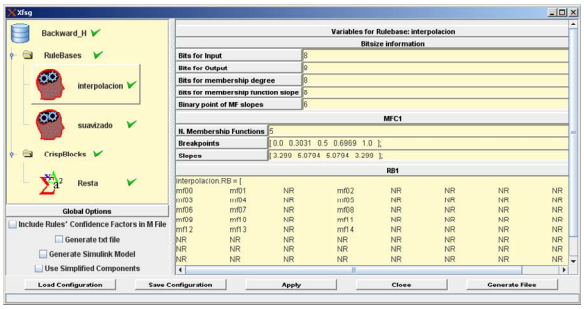



Figure 3. Main window of the tool *xfsg*.

(SysGen) tool in Matlab. Using the Xilinx Blockset in Simulink, a library of modules, named Xfuzzy Blockset or XfuzzyLib, has been developed to implement piecewise polynomial digital circuits accordingly to the previously commented active rule-driven architecture. These modules have masks that allow defining their parameters by Matlab command window or a configuration file. The tool *xfsg* generates automatically these configuration files. Some of the required parameters are obtained from the XFL3 descriptions (number of membership functions per input, knots of the B-splines and consequents of the rules). The other parameters related to bit size of the variables in the fuzzy and crisp modules should be introduced by the user through a graphical interface. Fig. 3 shows the window of *xfsg* corresponding to the hierarchical system already shown in Fig. 2.

In addition, *xfsg* can generate the Simulink model of the whole system. Hence, simulation performed in Xfuzzy can now be done in Simulink considering hardware details. Even the design can be synthesized and implemented in the FPGA and a hardware-in-a-loop simulation can be realized.

## IV. APPLICATION EXAMPLE

The design flow described above has been applied to develop an embedded controller for a car-like autonomous robot. The problem addressed has been to control the robot speed, $v$, and the angle of the front wheels that are responsible of the trajectory curvature, $\gamma$, so as to drive backward the robot from any configuration ($x$, $y$, $\phi$, $v$, $\gamma$) to an objective configuration (0, 0, 0, 0, 0) in the reference system shown in Fig. 4. The controller, therefore, has 2 outputs and up to 5 inputs. Instead of looking for a single rule base, which would be difficult to design, heuristic knowledge can be exploited to obtain a hierarchical system with 1- and 2-input modules. The

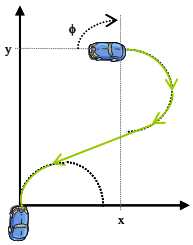


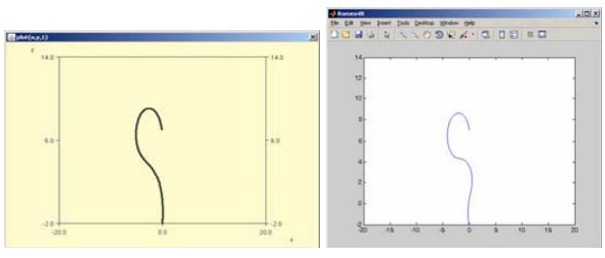

Figure 4. Navigation problem addressed.

first step is to separate speed and curvature control. Speed control depends on vertical position, *y*, and previous curvature. Curvature control depends on global position, *x*, *y*, and orientation, $\phi$. A second simplification is to further decompose curvature control into two modules connected in cascade. The first one evaluates, depending on *x* and *y*, which orientation, $\alpha$, should have the robot so as to drive backward with a zero curvature. The second module evaluates the difference between $\phi$ and $\alpha$ to decide the curvature. The structure of the curvature controller is that already shown in Fig. 2.

The 2-input module in charge of controlling speed is designed by translating linguistically expressed rules. Using the tools *xfplot*, *xfdm*, *xfsl*, and *xfsp*, this module is translated into a piecewise multilinear system with 4 B-spline functions covering *y* and 3 functions covering $\gamma$. Similarly, the 1-input module of the curvature controller is designed from linguistic rules and then translated into a piecewise linear system with 6 B-spline functions covering its input. The 2-input module of the curvature controller is designed from combining heuristic knowledge (expressed as linguistic IF-THEN rules) and geometrical analysis of the problem (expressed as input-output numerical data). Again using the tools *xfplot*, *xfdm*, *xfsl*, and *xfsp*, this module is designed as a piecewise multiquadratic system with 5 B-spline membership functions covering *x*, 3 functions covering *y*, and 15 consequents that depend linearly on the inputs.

Behavior of the designed controller working in a closed-loop with the robot model is verified with the tool *xfsim*. Fig. 5a shows one of these simulated trajectories.

Using the tool *xfsg*, configuration files are provided automatically to generate a Simulink model that uses modules of the Xfuzzy Blockset (which use, in turn, the modules in the Xilinx Blockset). Fig. 6 shows the Simulink model corresponding to the curvature controller. Hardware/software cosimulation of SysGen can be employed to implement the whole controller in a FPGA (in this case a Spartan3 xc3s700a of a Spartan 3A Starter Kit) and analyze its behaviour when working in a closed loop with a model of the robot (Fig. 7). Visualization utilities of Matlab allow obtaining the trajectories of the robot, like that in Fig. 5b. These results that consider hardware details may not meet control requirements. In those cases, the user can go back in the design flow and iterate with, for instance, different bit sizes of particular words. Since the whole design flow is automated, the user can move easily bottom to up and up to bottom.



(a) (b)

Figure 5. Simulation results in: (a) Xfuzzy 3, (b) Matlab.

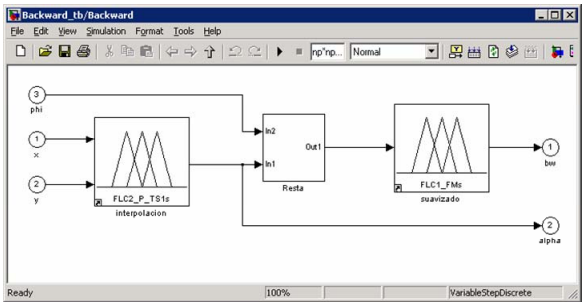



Figure 6. Simulink model of the curvature controller.

## V. Conclusions

The different CAD tools of Xfuzzy 3 are very helpful to pave the way between expressive linguistic models and efficient hardware implementations. Heuristic knowledge expressed linguistically and any other knowledge expressed numerically can be exploited by Xfuzzy to describe a hierarchical system made up of piecewise polynomial modules. Automated communication between Xfuzzy and Xilinx System Generator in Matlab allows the user to consider hardware details within the whole design flow.

## References


[1] A. M. Ibrahim, "Fuzzy logic for embedded systems applications", Elsevier Science, 2004.

[2] Xfuzzy web site: http://www.imse-cnm.csic.es/Xfuzzy.

[3] R. Rovatti, "Fuzzy piecewise multilinear and piecewise linear systems as universal approximators in Sobolev norms", IEEE Trans. on Fuzzy Systems, vol. 6 (2), pp. 235-249, 1998.

[4] R. Rovatti, "High speed implementation of piecewise-quadratic Takagi-Sugeno systems", Proc. IEEE International Fuzzy Systems Conf., pp. 292-297, 1999.

[5] S. Sánchez-Solano, A. Barriga, C. J. Jiménez, and J. L. Huertas, "Design and applications of digital fuzzy controllers," in Proc. IEEE Int. Conf. Fuzzy Syst., vol. 2, pp. 869–874, Jul. 1997.

[6] F.J. Moreno-Velo, I. Baturone, A. Barriga, S. Sánchez-Solano, "Automatic tuning of complex fuzzy systems with Xfuzzy", Fuzzy Sets and Systems, vol. 158, pp. 2026 – 2038, 2007.

[7] I. Baturone, F. J. Moreno-Velo, A. A. Gersnoviez, "A CAD approach to simplify fuzzy system descriptions", Proc. FUZZ-IEEE'2006, pp. 2392-2399, Vancouver (Canada), July 2006.

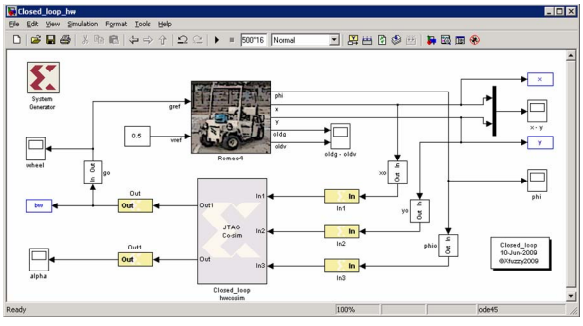




Figure 7. Hardware/software cosimulation in Matlab.