VLSI DESIGN OF UNIVERSAL APPROXIMATOR NEURO-FUZZY SYSTEMS

I. Baturone, S. Sánchez Solano, A. Barriga, C. J. Jiménez, R. Senhadji-Navarro, D. R. López.

Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica
Avda. Reina Mercedes s/n, (Edif. CICA),
E-41012, Sevilla, Spain

XVI Conference on Design of Circuits and Integrated Systems (DCIS2001), pp. 358-363, Oporto, Nov. 2001.

VLSI design of universal approximator neuro-fuzzy systems

I. Baturone, S. Sánchez Solano, A. Barriga, C. J. Jiménez, R. Senhadji-Navarro, D. R. López.

Instituto de Microelectrónica de Sevilla, I.M.S.E.-C.N.M.
Avda. Reina Mercedes s/n. Edificio CICA. 41012-Sevilla.
Phone: 95 505 66 66 FAX: 95 505 66 86
E-mail: lumi@imse.cnm.es

Abstract

Neuro-fuzzy systems can theoretically solve any problem since they are universal approximators. Besides, they combine the advantages of the neuro and fuzzy paradigms. This paper describes and compares the different strategies that can be adopted to implement the learning and inference mechanisms involved in a neuro-fuzzy system. CAD tools, most of them integrated into the fuzzy system development environment Xfuzzy 2.0 [1], have been developed to assist the designer in the implementation of neuro-fuzzy systems in FPGAs or ASICs.

1. Introduction

Neuro-fuzzy systems are drawing recently a great interest in many applications because they combine the advantages of the neuro and fuzzy paradigms: the automatic knowledge acquisition capability of neural networks and the human-like knowledge representation and reasoning capability of fuzzy systems. Besides, neuro-fuzzy systems are universal approximators and, since the processing involved is numerical, they can be implemented by microelectronic circuits. The design of an integrated circuit to implement a neuro-fuzzy system has to consider the requirements imposed not only by the fuzzy inference process but also by the learning process.

This paper focuses on discussing and comparing different microelectronic strategies for implementing universal approximator neuro-fuzzy systems, pointing out the advantages of using CAD tools. Section II describes the approximation capability of neuro-fuzzy systems. Section III discusses the two basic VLSI strategies for implementing the fuzzy inference process while Section IV describes the three strategies for implementing the neural learning process.

2. Neuro-Fuzzy Systems as Approximators

Among the fuzzy logic-based inference methods, the singleton or zero-order Takagi-Sugeno method is the most suitable for hardware implementation. In this method, the consequents of the fuzzy IF-THEN rules are represented by singleton values, c_i, and the output provided is given by:

$$y = \sum_{i=1}^{R} h_i \cdot c_i \Big/ \sum_{i=1}^{R} h_i \qquad (1)$$

where h_i is the activation degree of the i-th rule and R is the number of rules.

This behavior is equivalent to that of a neural network based on radial basis functions [2]. From this equivalence, it is apparent that the learning methods employed to tune radial basis function networks can be applied to tune singleton fuzzy systems. We will name these so-tuned singleton fuzzy systems as singleton neuro-fuzzy systems.

Let us consider a singleton neuro-fuzzy system whose input universes of discourse are partitioned into L linguistic labels. If α is the maximum overlapping degree, L+1-α intervals can be distinguished per input. They are separated by the points $\{a_{i1}, a_{i2}, ..., a_{iN}\}$, where N is L-$\alpha$. Given an input vector $\bar{x}_o=(x_{1o}, ..., x_{uo})$, the output provided by the neuro-fuzzy system depends only on α^u active rules:

$$y(\bar{x}_o) = \left. \frac{\sum_{k=1}^{\alpha^u} h_{pk}(\bar{x}) \cdot c_{pk}(\bar{x})}{\sum_{k=1}^{\alpha^u} h_{pk}(\bar{x})} \right|_{\bar{x}=\bar{x}_o} = y_p(\bar{x})\big|_{\bar{x}=\bar{x}_o} \qquad (2)$$

where h_{pk} is the activation degree of one of the α^u active rules and c_{pk} is its corresponding consequent function.

This work has been partially supported by the Spanish CICYT Project TIC98-0869 and the FEDER Project 1FD97-0956-C3-02.

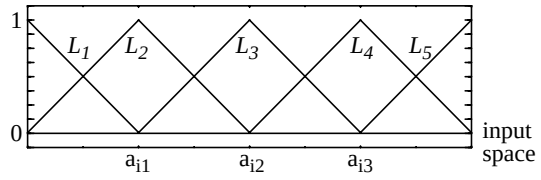

Fig. 1: Covering of the input space with an overlapping degree of 2.

The parameters that define the piece $y_p(\bar{x})$ are a few of the global parameters that define the global output, $y(\bar{x})$. In this sense, a neuro-fuzzy system is viewed as a local piecewise interpolator. The interpolation provided can be piecewise constant, piecewise linear, piecewise quadratic, or piecewise nonlinear, in general, depending on whether y_p is constant, linear, quadratic, or nonlinear in $\bar{x}$. In particular, neuro-fuzzy systems that cover their input universes of discourse with normalized triangular membership functions (which are B-splines of degree 1), as shown in Figure 1, and that employ the normalized product or the meet-like operator as antecedent connective, are piecewise multilinear interpolators [3]-[4]. Those neuro-fuzzy systems which employ B-splines of degree 2 and the product or the meet-like operator as antecedent connective are piecewise multiquadratic interpolators. In general, the use of nonlinear membership functions or the minimum as connective lead to piecewise nonlinear interpolators. In any case, neuro-fuzzy systems can approximate any input-output surface and that is why they can be called universal approximators [3]-[4].

3. VLSI strategies for the inference mechanism

There are basically two VLSI strategies for implementing the fuzzy logic-based inference of a neuro-fuzzy system. One of them, which will be called memory-based approach, is to implement directly each piece y_p of the interpolation. This approach has been employed in digital realizations with DSPs [4] and analog ASICs [5]. Programmable parameters of this approach can be the points a_{ij} that allow identifying the active grid cell, and the constants that define the piece y_p in that grid cell. The other strategy is to implement the different stages of the fuzzy inference mechanism. These stages are: calculation of the membership degrees, $\mu_j(x_i)$, (by membership function circuits, MFCs), calculation of the activation degrees of the rules, $h_{pk}(\bar{x})$, (by connective circuits), and computation of a weighted average. This approach will be called MFC-based approach because its main difference with the previous one is that membership degrees are calculated explicitly. This approach has been followed by many analog, digital and mixed-signal fuzzy ASICs [6]. The programmable parameters can be again the points a_{ij} (which separate the L_i+1-α intervals) and the parameters of the consequent functions c_{pk}. Universal approximator fuzzy chips can be obtained by both approaches. They can be adjusted to a particular application by an adequate learning mechanism. These two approaches for implementing piecewise interpolator systems are illustrated in the following.

3.1. Memory-based approach

The most direct strategy to approximate a given surface is to implement a look-up table that contains all the output values corresponding to every possible combination of the input signals. If there are *u* inputs, $\bar{x}=(x_1, ..., x_u)$, coded by *N* bits, a look-up table divides the input space into $2^{N \cdot u}$ cells and associates to each cell a constant output value. Hence, look-up tables implement **piecewise constant interpolators**.

The microelectronic circuits suitable to carry out this strategy are standard digital memories and programmable logic devices (PLDs). One advantage of this realization is that the response time is very short because the only operation (apart from the possible data conversions) is to retrieve data from a memory and this can be performed in a few nanoseconds. As a drawback, look-up table implementations are practical only when the number of inputs and their quantization degrees are low, otherwise the total number of bits to store is very large.

The fuzzy system development environment Xfuzzy [1] has a tool named *xftl* which automatically directs the synthesis tools of Synopsys and Xilinx to obtain an implementation of a piecewise constant interpolator on an FPGA. The input to *xftl* is the description of the interpolator system in the high-level specification language of Xfuzzy: XFL (this description can be performed via graphical user interfaces). The output of *xftl* consists of two files. The first file contains a formal description of the look-up table in PLA format. This table can be minimized by any tool able to accept the PLA format and implemented either by a combinational circuit or by a memory. The second of the output files of *xftl* is intended to control the Synopsys logical synthesis tools for minimizing the look-up table and extracting the Boolean equations in order to implement the table using the combinational resources of a Xilinx FPGA. The graphical user interface of *xftl* is shown in Figure 2.

Another way to approximate a given surface is to divide the input space into $2^{N \cdot u}$ cells and associate to each cell a multilinear function, $y= y_p(\bar{x})$. Compared with look-up tables, **piecewise multilinear interpolators** need less input space resolution. In the case of two inputs, they provide the following kind of pieces:

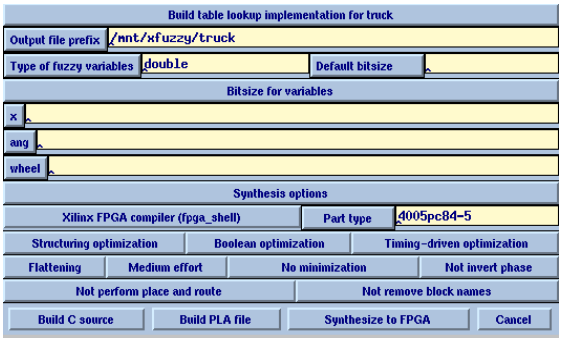

Fig. 2: Graphical user interface for *xftl*.

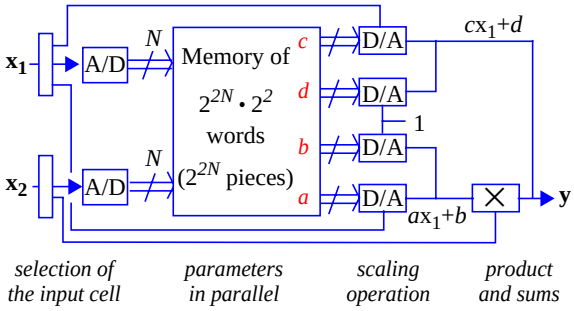

Fig. 3: Memory-based parallel current-mode realization of a piecewise multilinear controller.

$$y_p(x_1, x_2) = ax_1x_2 + bx_2 + cx_1 + d \qquad (3)$$

where *a*, *b*, *c*, *d* are constants.

The memory-based implementation of programmable piecewise multilinear controllers requires to retrieve the programmable parameters from a memory and to implement multiplication and addition operations.

A parallel realization of these operations offers the shortest processing time but the largest circuitry. For instance, the memory that contains the programmable parameters (*a*, ..., *d*, in Equation (4)) must have 2^u output buses. In the case of analog current-mode circuitry, the additions can be performed by simply connecting wires; and the following blocks are required: D/A converters (like programmable current mirrors), to convert and to scale the constants; and multipliers. The block diagram of this type of realization for the case of two inputs is shown in Figure 3.

The circuitry can be reduced by performing the operations sequentially. For instance, operations can be performed in 2^u steps by using 1 adder, 1 multiplier, and a memory with 1 output bus, as shown in Figure 4 for the case of two inputs.

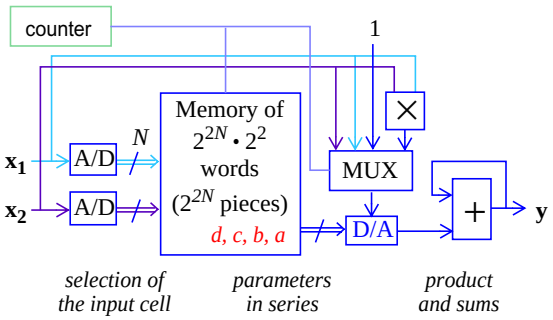

Fig. 4: Memory-based sequential realization of a piecewise multilinear interpolator.

3.2. MFC-based approach

Instead of implementing directly the expression of each output piece, the MFC-based approach implements the different stages of a fuzzy inference mechanism: fuzzification, calculation of the activation degrees of the rules and computation of a weighted average. Circuitry can be simplified if certain constraints are imposed. In this sense, the membership functions that cover each input universe of discourse usually overlap each other with a degree, α, of 2 or 3. This happens, for instance, when B-splines of degrees 1 and 2 are selected as membership functions. This means that 2 or 3 MFCs per input are required to generate the membership functions. Another constraint is that the membership functions can be normalized, that is, their sum is constant (as shown in Figure 1). In this case, α-1 MFCs per input are required because the other membership degree can be calculated with an addition.

If normalized membership functions are employed and if the normalized product or the meet-like operator is chosen as the connective operator, the denominator of the weighted average is constant, so that no divider is required.

If the membership functions do not overlap, the output value is directly the consequent function and there is anything fuzzy. In this case and if the consequents are constant, the result is a **piecewise constant interpolator**.

If the overlapping degree of membership functions is 2, the input space is divided into $(2^M\text{-}1)^u$ cells, where 2^M is the number of membership functions that cover each input variable. If the membership functions are linear (like those in Figure 1), the antecedent connective is the normalized product or the meet-like operator, and the consequents are constant, the resulting fuzzy system is a **piecewise multilinear interpolator**. For instance, given two inputs, x_1 and x_2, and using the normalized product as antecedent connective the output piece is given by:

$$y_p(x_1, x_2) = c_{RR}\mu_{R1}\mu_{R2} + c_{RL}(\mu_{R1} - \mu_{R1}\mu_{R2}) +$$

$$c_{LL}(1 - \mu_{R1} - \mu_{R2} + \mu_{R1}\mu_{R2}) + c_{LR}(\mu_{R2} - \mu_{R1}\mu_{R2}) \qquad (4)$$

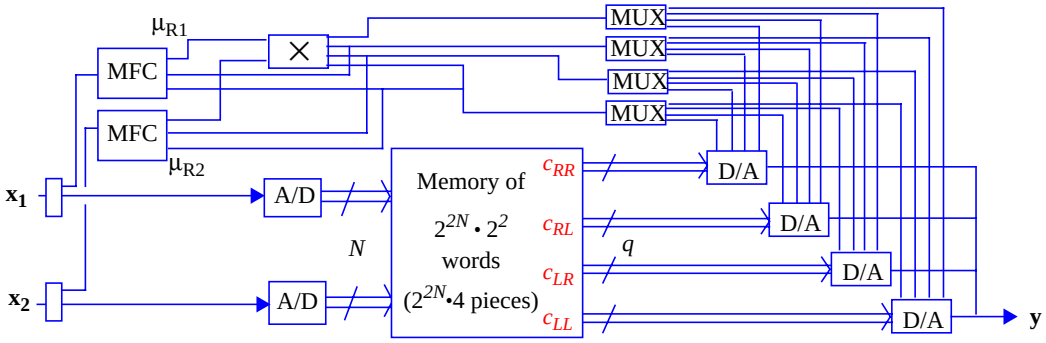

Fig. 5: MFC-based parallel current-mode realization of a piecewise multilinear fuzzy system with $4 \cdot 2^{2N}$ rules.

where μ_{Ri} and $1-\mu_{Ri}$ are the two non-zero membership degrees of the input x_i.

If the minimum operator is chosen as the antecedent connective, the denominator of the weighted average is not constant, a divider is required, and a **piecewise nonlinear interpolation** is performed.

Similar to the memory-based approach, this MFC-based approach requires also multiplications between two variables and to retrieve 2^u constants from a memory. An important difference is that the number of words to store is now $2^{M \cdot u}$, while it is $2^{(L+1) \cdot u}$ in the memory-based approach, considering input space partitions of $(2^M-1)^u$ and $2^{L \cdot u}$ cells, respectively. This means that, for equal input space partitions, the memory size in the MFC-based approach is almost half of the memory size in the other approach. The reason is that each word of the memory in the MFC-based strategy is used by 2^u different cells while in the other approach it is used by only 1 cell. The cost to pay for this memory reduction is to include several multiplexers and, in this case, 1 MFC block per input variable.

A parallel current-mode realization following this MFC-based strategy is shown in Figure 5 for the case of a piecewise bilinear interpolation. In a parallel realization, the memory of $2^{M \cdot u}$ words is divided into 2^u parts, each one containing $2^{(M-1)u}$ words that are never required simultaneously. Multiplexers have to be added to select which of the 2^u parts provides each of the 2^u words required [7]. The circuitry can be reduced by sequentializing the operations. For instance, operations can be performed in 2^u steps by using 1 accumulator, 1 multiplier, and a memory with 1 output bus [6].

There are three basic approaches to implement the MFCs. One approach is to generate all the membership functions of a variable by means of a memory. This is the most versatile approach because it allows different shapes for all the membership functions (bell-shaped, triangular, trapezoidal, etc.). A second approach is to generate the same shape for all the membership functions of an input variable by means of a memory (in this case, the first operation is to identify the input interval to which the input belongs). The third approach is to generate the same shape for all the membership functions of an input variable by means of an arithmetic block (again the input interval to which the input belongs has to be identified firstly). Examples of arithmetic MFCs are current-mode analog MFCs or digital MFCs that generate triangular membership functions and transconductance-mode analog MFCs, based on differential pairs, that generate bell-shaped membership functions [6].

The environment Xfuzzy has a tool named *xfvhdl* which starts from the high-level specification of the fuzzy system and produces a VHDL description. This tool is intended to automatically synthesizing piecewise nonlinear interpolators (with a minimum operator as antecedent connective) that follows a sequential architecture and that implement the MFCs with memories or with arithmetic circuits. The code generated by *xfvhdl* can be used for implementing the fuzzy system as an ASIC or as a Xilinx FPGA. The graphical user interface of *xfvhdl* is shown in Figure 6.

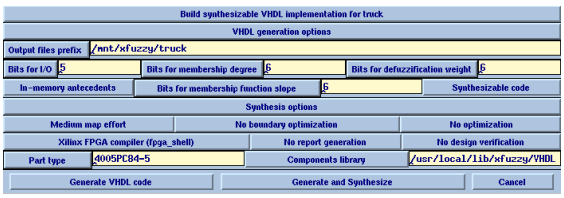

Fig. 6: Graphical user interface for *xfvhdl*.

4. Strategies for the learning mechanism

Typical neural learning mechanisms are supervised, that is, they employ a set of numerical input values and their desired output values, and the objective is to minimize the error between the actual and the desired output for the given input vectors.

4.1. Off-chip learning

Two factors are determinative when selecting an adequate strategy to implement the learning mechanism. One factor is the type of application towards which the neuro-fuzzy system is addressed. Static applications, like the modeling of static systems, employ fuzzy systems whose knowledge base does not change with time. Hence, the learning process can be performed **off chip** and only the inference process needs to be integrated on silicon. In this case, a second factor to be considered is the application range of the circuit. If it is conceived to be dedicated to a specific application, the learning process can be performed prior to the fabrication of the chip and the learnt fuzzy system can be integrated in a non-programmable device, which is the least costly solution. If the circuit is going to be applied to different tasks, it should be programmable. In this case, the learning process is carried out prior to the use of the chip and the learnt parameters are downloaded to the chip before its operation. A precise emulation of the fuzzy inference hardware is very important for the effectiveness of the off-chip learning. For this purpose, we have developed a CAD tool named *fmcsim* that simulates at a behavioral level the performance of the fuzzy inference hardware. Given a desired input-output mapping to reproduce, this tools allows: (a) to compare the output provided by the different VLSI strategies for the inference mechanism depending on the input partitions, the resolution of the building blocks employed (memories and computing blocks), the types of MFCs and connective operators (in the case of an MFC-based approach), and (b) to find the adequate input partitions and the adequate values for the parameters that define the output pieces, taking into account the resolution of the building blocks.

As an example, Figure 7 illustrates the results obtained with *fmcsim* when off-chip adjusting different designs of a fuzzy inference chip to approximate the following surface (Figure 7a):

$$z = 0.5(1 + \sin(2\pi x)\cos(2\pi y)) \quad (5)$$

The fuzzy inference chips simulated follow an MFC-based approach, implement 3x5=15 rules, and employ consequents with an 8-bit resolution. Figure 7b shows the simulated output provided by a piecewise linear digital realization which employs the meet-like operator as connective and MFCs implemented by memories with words of 3 bits. Figure 7c illustrates the output of an analog piecewise nonlinear realization which employs the product as connective and transconductance-mode MFCs working in strong inversion. In this example, it can be seen that digital computing blocks introduce quantization errors that usually dominate the approximation error.

4.2. On-chip learning

There is another type of applications for which a dynamically changing knowledge base is needed. Examples of this type of applications are identification and modeling of non-linear dynamic systems, adaptive prediction of time series, adaptive noise cancellation, adaptive control, etc. In these cases, the neuro-fuzzy systems employed are called adaptive fuzzy systems because they adapt their knowledge base to match the new situation, thus resorting to the use of the learning mechanism as they are operating. Two approaches can be used for these applications. One of them is to implement the learning mechanism **on chip** together with the inference mechanism. This is the feasible solution when requirements of low power consumption, embedded operation (low area occupation), and real-time performance apply not only to the inference engine but also to the learning loop [6].

The tool *fmcsim* also allows the user to emulate the behavior of a neuro-fuzzy chip with on-chip learning mechanisms: weight perturbation for antecedent parameters and gradient-descent for the singleton values of the consequents.

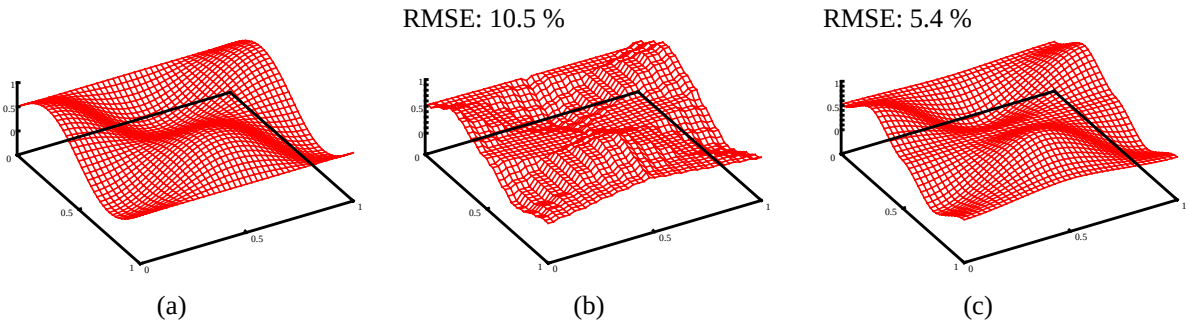

Fig. 7: Simulated performance of different VLSI neuro-fuzzy systems approximating the surface shown in (a).

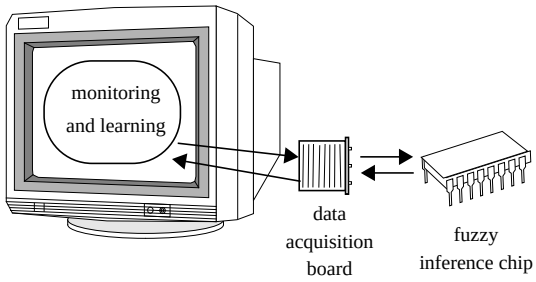

Fig. 8: Block diagram of chip-in-the-loop learning.

4.3. Chip-in-the-loop learning

The third approach to implement the learning mechanism is known as **chip-in-the-loop** training and can be applied if the chip is programmable. This approach relies on a host computer which provides the input training vectors to the chip, compares the desired output vectors with the output generated by the chip, and reprogram adequately the chip. Hence, the computer must be provided with a data acquisition card, as shown in Figure 8.

A CAD tool named *xflab* allows the designer configuring the data acquisition card, defining the input and output signals, and monitoring the fuzzy inference chip [6]. Figure 9 shows the main window of the *xflab* graphical user interface as well as the input/output configuration window.

5. Conclusions

Different strategies for implementing the inference and learning mechanisms of universal approximator neuro-fuzzy systems have been described and compared. The VLSI strategies for the inference mechanism differ in the type of hardware resources employed (more use of memory against computing blocks or vice versa) and in the system performance achieved (more flexibility against simplicity or vice versa). Piecewise constant, linear, and non-linear behavior have been discussed, and analog, digital, and mixed-signal realizations have been illustrated. The adequate implementation strategy for the learning mechanism depends on the type of application. For off-chip or chip-in-the-loop learning strategies, the learning mechanism is implemented by a host computer. Different CAD tools (*xftl*, *xfvhdl*, *fmcsim*, and *xflab*), most of them currently integrated into the Xfuzzy 2.0 development environment, have been developed to assist the designer in the hardware implementation of neuro-fuzzy systems as FPGAs or ASICs.

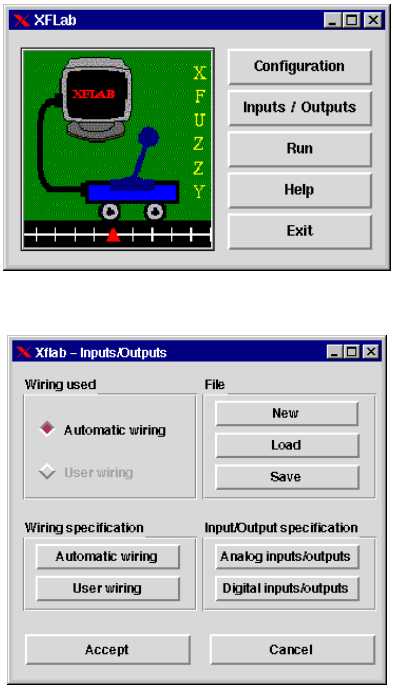

Fig. 9: Graphical user interface for *xflab*

References

[1] D. R. López, C. J. Jiménez, I. Baturone, A. Barriga, S. Sanchez-Solano, “Xfuzzy: a design environment for fuzzy systems”, *Proc. 7th Int. Conf. on Fuzzy Systems*, pp. 1060-1065, Anchorage, May 1998.

[2] J.-S. R. Jang, C.-T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems", *IEEE Trans. Neural Networks*, vol. 4 (1), pp. 156-159, 1993.

[3] X.-J. Zeng, M. G. Singh, “Approximation accuracy analysis of fuzzy systems as function approximators”, IEEE Trans. on Fuzzy Systems, vol. 4 (1), pp. 44-63, Feb. 1996.

[4] Rovatti, R., Vittuari, M., Linear and fuzzy piecewise-linear signal processing with an extended DSP architecture, in *Proc. IEEE Int. Conference on Fuzzy Systems*, pp. 1082-1087, Anchorage, 1998.

[5] Matas, J., García de Vicuña, L., Castilla, M., A synthesis of fuzzy control surfaces in CMOS technologies, in *Proc. IEEE Int. Conference on Fuzzy Systems*, vol. 2, pp. 641-647, Barcelona, 1997.

[6] Baturone, I., Barriga, A., Sánchez-Solano, S., Jiménez-Fernández, C. J., López, D. R., *Microelectronic design of fuzzy logic-based systems*, CRC Press, 2000.

[7] Baturone, I., Barriga, A., Sánchez-Solano, S., Huertas, J. L., “Mixed-Signal Design of a Fully Parallel Fuzzy Processor”, *Electronics Letters*, vol. 34 (5), pp. 437-438, March 1998.