VHDL HIGH LEVEL MODELLING AND IMPLEMENTATION OF FUZZY SYSTEMS

Barriga A ¹, Sánchez-Solano¹ S, Brox P¹, Cabrera A ², Marbán M.A

¹ Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica
Avda. Reina Mercedes s/n, (Edif. CICA)
E-41012, Sevilla, Spain

² Dpto. Automática y Computación. Facultad de Ingeniería Eléctrica. Instituto Superior Politécnico “José Antonio Echeverría”, Ciudad de la Habana, Cuba.

***Proc. International Workshop on Fuzzy Logic and Applications (WILF'2003),
pp. xx-xx, Nápoles, Oct. 2003.***

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

VHDL High Level Modelling and Implementation of Fuzzy Systems ¹

Barriga A ¹, Sánchez-Solano¹ S, Brox P¹, Cabrera A ², Baturone I¹

¹ Instituto de Microelectrónica de Sevilla, España,
barriga@imse.cnm.es

² Instituto Superior Politécnico José Antonio Echeverría (ISPJAE), Ciudad de La Habana, Cuba

Abstract. In this paper we illustrate a fuzzy logic system design strategy based on a high level description. Employing this high level description, the knowledge base is described in a language in appearance close to the natural language with the particularity that it uses a hardware description language (VHDL) directly synthesizable on an FPGA circuit. In addition, we analyze FPGA implementations of different fuzzy inference hardware architectures in order to characterize them in terms of area and speed.

1 Introduction

Natural language uncertainty and the approximate reasoning mechanism of the human brain can be modelled through the use of fuzzy logic. The knowledge base of a fuzzy system is described by a set of rules as in an expert system. However, the rule computation is performed numerically as in a neural network. This double view (symbolic description and analytic processing) means that fuzzy logic is useful for modelling those systems in which it is difficult to achieve a mathematical behaviour.

Electronic implementation of fuzzy systems is necessary to create specific hardware at a low cost but with a high performance, in terms of speed and power. Thus, our design methodology has to cover the different design stages, from system specification up to the implementation and testing of the circuit. Our design methodology is based on the fuzzy system development environment called Xfuzzy [1], [2]. The tools included in this environment share a fuzzy system description language (XFL [3]) for the rule base and structure specification. Xfuzzy includes tools that make the description of fuzzy system easier. There are also simulation tools and learning tools for adjusting the system parameters. The realization of the system can be achieved by using the synthesis tools, both software (C, C++ and Java) or hardware synthesis. Hardware synthesis is performed on a specific architecture of the fuzzy system [4], [5], [6]. This architecture exploits the active rule concept, programmability, and uses simplified defuzzification methods to optimize the implementation. In order to avoid the restrictions imposed by the architecture, it is useful for some applications to model the fuzzy system with VHDL [7] while the synthesis tools perform the implementation of the circuit. The aim of our paper is to depict a fuzzy logic modelling. We will show a fuzzy logic modelling style based on VHDL starting from a high level description and the implementation on FPGAs devices.

2 VHDL System Modelling

In this paper we achieve two aims. First we propose a fuzzy system description style based on the description of the structure of the fuzzy system in an easy way (linguistic variables, rule base, fuzzy operators). Secondly, we analyze a set of hardware architectures in order to characterize them in terms of speed and area. Our starting premise implies that the fuzzy system description must be synthesizable because we are interested in hardware realizations. Thus we will use VHDL as the working platform for our systems. But VHDL language imposes some limitations, compared with the flexibility and expressiveness of other fuzzy logic oriented languages (such as XFL [3]). On the other hand, it is necessary to adapt the characteristics of the system (types of membership functions, inference algorithms, defuzzification mechanisms) to its hardware implementation.

To achieve the first objective we use the XFL description format. In this language the system structure description (fuzzy sets, rule base) and the operator description (connectives, fuzzy operations) are defined separately. This allows us to describe independently both the fuzzy system structure and the processing algorithm.

¹ Partial founded by projects 58/02 de DGAECE (Junta de Andalucía) and TIC2001-1726 (CICYT).

2.1 Rulebase

Figure 1 shows an example of the VHDL architecture body of a fuzzy system. It contains the rule base structure. In this example there are 9 rules. Each rule structure can be divided into two components: the antecedent of the rule and the consequent. The antecedent of the rule is an expression of the input variables related to their linguistic values. The consequent sets the linguistic value of the rule output.

The rule in Figure 1 must be read as follows:

$$\text{if } x \text{ is } \mathbf{low} \text{ and } y \text{ is } \mathbf{low} \text{ then } z \text{ is } \mathbf{half} \quad (1)$$

where x, y, z are the input and output variables, **low** and **half** are linguistic labels.

```
architecture specification of example1 is
    . . .
begin
    R(1) <= rule( (x = low)      and (y = low),      half);
    R(2) <= rule( (x = low)      and (y = middle),   strong);
    R(3) <= rule( (x = low)      and (y = high),     very_strong);
    R(4) <= rule( (x = middle)   and (y = low),      weak);
    R(5) <= rule( (x = middle)   and (y = middle),   half);
    R(6) <= rule( (x = middle)   and (y = high),     strong);
    R(7) <= rule( (x = high)     and (y = low),      very_weak);
    R(8) <= rule( (x = high)     and (y = middle),   weak);
    R(9) <= rule( (x = high)     and (y = high),     half);
    Z <= defuzz(R);
end specification;
```

Fig. 1. VHDL description of the rulebase.

The processing mechanism of the fuzzy operation *is* (=), connective *and*, inference *then* (*rule*(,)) is not defined in the VHDL description. Only the structure of the rulebase is defined. Such a description is a high level description because it does not assume any specific implementation criteria. It only describes the knowledge base in terms of a behaviour rule base.

Figure 2 shows an example of a rulebase in which rules are grouped using the connective *or* and the linguistic hedges *greater than* (>) and *less than* (<). Using these kinds of operators we can reduce the rulebase. The equivalent rulebase of Figure 1 will thus contain only five rules instead of nine.

```
architecture specification of example2 is
    . . .
begin
    R(1) <= rule(((x = middle_low) and (y = very_low)) or
                ((x = middle)      and (y < middle_low)) or
                ((x = middle_high) and (y < middle)) or
                ((x = high)         and (y < middle_high)) or
                ((x = very_high)    and (y < high))), very_weak);
    R(2) <= rule(((x = very_low)   and (y > low)) or
                ((x = low)         and (y > middle_low)) or
                ((x = middle_low)  and (y > middle)) or
                ((x = middle)      and (y > middle_high)) or
                ((x = middle_high) and (y = very_high))), very_strong);
    . . .
end specification;
```

Fig. 2. Rulebase using linguistic hedges.

2.2 Membership Function

Linguistic labels represent a range of values within the universe of discourse of input and output variables. These labels can be described by functions in order to compute the membership degree of a certain input value. Membership functions associated to a linguistic label can be triangular or trapezoidal. Figure 3 shows the definition of such membership functions.

```

architecture specification of example1 is
  -- MF for x and y
  constant low   : trapezoid := (0, 0, 15, 31);
  constant middle: triangle  := (15, 31, 47);
  constant high  : trapezoid := (31, 47, 63, 63);
  --MF for z
  constant very_weak : integer := 0;
  constant weak      : integer := 15;
  constant half      : integer := 31;
  constant strong    : integer := 47;
  constant very_strong: integer := 63;
  . . .
begin
  . . .
end specification;

```

Fig. 3. VHDL membership function definitions.

A triangular or trapezoidal membership function is characterized by a name (linguistic label) and three points defining the triangle (or four points defining the trapezoid). In the example of Figure 3 there are three constants related to the membership functions of the input variables x and y (low, middle and high) and five singleton membership functions of output variable z (very_weak, weak, half, strong and very_strong). Figure 4 shows these function sets.

Fig. 4. Membership function representation.

2.3 Data Types and Fuzzy Operators

Data types and fuzzy operators are defined independently of the system description. They are inserted in a VHDL package as a type and function library. Fuzzy operators are defined using overloaded functions.

```

package xfvhdfunc is
  type triangle is array (1 to 3) of integer range 0 to TOP;
  type trapezoid is array (1 to 4) of integer range 0 to TOP;
  function "=" (x: integer; y: triangle) return integer;
  function ">" (x: integer; y: triangle) return integer;
  function ">=" (x: integer; y: triangle) return integer;
  function "<" (x: integer; y: triangle) return integer;
  function "<=" (x: integer; y: triangle) return integer;
  function "and" (x,y: integer) return integer;
  function "or" (x,y: integer) return integer;
  function "not" (x: integer) return integer;
  function rule (x: integer; y: integer) return two_integer_vector;
  function defuzz(x: consec) return two_integer_vector;
end xfvhdfunc;

```

Fig. 5. VHDL package containing data type definitions and functions.

Figure 5 shows the VHDL package containing the definition of some data types and functions. Most of the functions overload VHDL operators. We can get compact specification of the rulebase using overloaded operators, as shown in Figure 1.

Another characteristic of the function library is the definition of linguistic hedges (greater, greater or equal, less, less or equal). We can get more linguistic expressiveness and compact rulebases using these operators, and as result we get a better optimization of the resulting circuit (this will be discussed in Section 4).

3 Fuzzy System Architectures

The set of fuzzy system architectures we have considered is shown in Table 1. The two first architectures share the same entity (input and output ports). Both are combinational circuits implementing the rulebase in parallel. An example of the first architecture is the rulebase of Figure 1. In this example there are 9 rules executing concurrently. As a result of the synthesis process, the final circuit computes all the rules simultaneously.

The second architecture of Table 1 is also implemented as a combinational circuit. The main difference between the previous architecture and this one is the use of the connective *or* and the linguistic hedges, as shown in Figure 2. As mentioned in a previous section, these operators were implemented using overhead VHDL operators.

The resulting synthesis of both architectures produces very similar circuits. The main difference is related to the number of rules (Architecture 2 has only 5 rules, one for each consequent). Thus, the antecedent part of the rules can be optimized better in Architecture 2, and as a result, the final circuit requires less hardware resources.

Architecture		Entity	
1	Combinational system with parallel rule processing		z
2	Combinational system with parallel rule processing, rule aggregation and linguistic hedges		
3	Sequential rule processing		z

Table 1. Architectures being considered.

Architecture 3 is based on sequential execution of the rulebase. The description uses a finite state machine with a machine state for each rule. The entity includes a start signal (input capture, machine initialization) and a finish signal.

4 Implementation Results

As an implementation example we will consider the following function:

$$z = \frac{1}{1 + e^{10(x-y)}} \quad (2)$$

It consists of a two-input function (x, y). The surface representing the function is shown in Figure 6. We have analyzed two set of realizations. A first set is composed of systems containing 3 membership functions (3 MF) for the input variables and 5 membership functions for the output. We have shown along this paper some examples of these kinds of systems. Membership functions were described in Figures 3 and 4. The rulebase is described in Figure 1 and we have considered different realizations according to Table 1. All cases use 6 bits for input variables, membership degree and slopes.

The right surface of Figure 6 shows an approximation using 7 membership functions (7 MF) for antecedents and 5 for consequents. Both kind of approximations allow us to compare realizations of less complex systems (case using 3 MF) and more complex systems for more precision (case 7 MF). The rulebase for Architecture 1 using 3 MF requires 9 rules while using 7 MF requires 49 rules. We have implemented all the systems on Xilinx FPGAs (4000 and Spartan family devices).

Fig. 6. Function surface (Equation 2) using 3 and 7 membership functions for antecedents.

Comparing Architectures 1 and 2 of Table 1 we have observed a reduction in terms of hardware resources and maximum delay when we reduce the rule number using the connective *or* and linguistic hedges. Such a reduc-

tion means about 25% for a 3 MF and about 56% for a 7 MF. The reason is that in Architecture 2 the aggregation rule stage is simpler because the aggregation is implemented summing the outputs in parallel by means of a combinational circuit.

Regarding speed the best choice is Architecture 2 in order to optimize the maximum delay. As an example we will show some implementation results on Xilinx 4005XL FPGA. The maximum delay for Architecture 1 were 50 nsec. and 170 nsec. using 3 MF and 7 MF, respectively. The delays for Architecture 2 were 12 nsec. and 100nsec. using 3 and 7 MF, respectively.

Architecture 3 means a sequential execution of rules. The realization based on this architecture increases hardware costs, compared with Architecture 2, about 170% for less complex systems (3 MF). However, it shows better results for complex systems (7 MF) requiring about 1% less resources than Architecture 2. The reason is related to the overhead of the finite state machine which is most significant for less complex systems. In more complex systems such an overhead is compensated by the cost reduction of the defuzzification stage.

Concerning speed, the maximum delays of Architecture 2 and 3 are similar. Nevertheless, circuits based on Architecture 2 require one clock cycle per inference while the circuits based on Architecture 3 require one clock cycle per rule to perform an inference.

As a final conclusion, Architecture 2 (description shown in Figure 2) provides better implementation results in terms of area and speed. Such systems have a more compact rulebase which allows for the optimization of the output stage of the circuit (rule aggregation and defuzzifier). We can thus conclude that one of the main design criteria for fuzzy system implementation is the reduction of the rulebase.

We are currently developing a CAD tool under the Xfuzzy [2] fuzzy system development environment which produces the VHDL description of the XFL specification.

5 Conclusions

High level description of fuzzy systems allows the designer to focus on the structure of the system and the behaviour specification and not on the implementation features of the processing functions. The use of a hardware description language with a predefined function library has the advantage of a direct approach to the circuit through a synthesis tool. FPGA implementations allow us to run the design cycle automatically in order to obtain the circuit in a short time.

In this paper we have shown the modelling of fuzzy systems using VHDL and we have analyzed various architectures. This study is the base for developing tools in order to automate the design and select the best choice for a specific problem.

References

1. F.J. Moreno Velo, I. Baturone, S. Sánchez Solano, A. Barriga: "XFUZZY 3.0: A development environment for fuzzy systems". International Conference in Fuzzy Logic and Technology (EUSFLAT'2001), pp. 93-96, Leicester, Sept. 2001.
2. Xfuzzy home page: <http://www.imse.cnm.es/Xfuzzy>
3. F.J. Moreno Velo, S. Sánchez Solano, A. Barriga, I. Baturone, D.R. López: "XFL3: A new fuzzy system specification language". WSES/IEEE Multiconference on Circuits, Systems, Communications and Computers (CSCC'2001), pp. 361-366, Creta July 2001.
4. Hollstein, T., Halgamuge, S.K., Glesner, M.: "Computer-Aided Design of Fuzzy Systems Based on Generic VHDL Specifications". IEEE Trans. on Fuzzy Systems, vol. 4, no. 4, Nov. 1996.
5. Sánchez-Solano, S., Barriga-Barros, A., Jiménez, C.J., Huertas, J.L.: "Design and Application of Digital Fuzzy Controllers". 6th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'97), pp.869-874, Barcelona, 1-5 July. 1997.
6. S. Sánchez-Solano, R. Senhadji, A. Cabrera, I. Baturone, C.J. Jiménez, A. Barriga: "Prototyping of fuzzy logic-based controllers using standard FPGA development boards". Proc. 13th IEEE Int. Workshop on Rapid System Prototyping (RSP'2002), Darmstadt, Jul. 2002.
7. Zamfirescu, A., Ussery, C.: "VHDL and Fuzzy Logic If-Then Rules". Proc. of Euro-VHDL, pp. 636-641, Hamburg 1992.