Design and Simulation of the Type-2 Fuzzification Stage: Using Active Membership Functions

Oscar Montiel[1], Roberto Sepúlveda[1], Yazmín Maldonado[2], and Oscar Castillo[3]

[1] Centro de Investigación y Desarrollo de Tecnología Digital del Instituto Politécnico Nacional (CITEDI-IPN), Av. del Parque No.1310, Mesa de Otay, 22510, Tijuana, B.C., México
o.montiel@ieee.org,r.sepulveda@ieee.org

[2] M.S. Student at CITEDI-IPN
maldonado@citedi.mx

[3] Division of Graduate Studies and Research, Calzada Tecnológico S/N, Tijuana, B.C., México
ocastillo@hafsamx.org

Abstract. This paper describes the design and implementation of the fuzzification stage for type-1 and type-2 fuzzy inference systems (FIS). A versatile method to calculate the membership values was used, it handles real numbers using decimal floating point binary encoding to calculate the slopes of triangular and trapezoidal membership functions. The designs were developed using VHDL code for FPGA implementation. The type-1 implementation is shown to give the basis of the type-2 implementation, which is based on the average method that consists in substituting an interval type-2 FIS by two type-1 FISs to cope with uncertainty. The functionality of the designs were evaluated by the analysis of the control surface plots of a speed controller for a DC motor. The plots were obtained from Simulink models that includes the VHDL designs developed in the Xilinx ISE. They were imported to the Simulink environment through the Xilinx System Generator.

1 Introduction

Recently, there has been an increasing interest in the research and implementations of type-2 fuzzy systems because they offer bigger advantages in handling uncertainty with respect to type-1 fuzzy systems [4, 7, 9, 10, 11]; so, many researchers have found this characteristic very interesting to be applied in the design of new digital controllers. This statement is supported by several research reports that cover from theoretical studies to physical implementations [20, 21]. However, it is well known that the implementation of type-2 fuzzy systems demand higher computational resources, this requirement is very difficult to satisfy for low cost microcontroller systems for industrial or consumer use.

With the rapid development of the Very Large Scale Integrated Circuits (VLSI), such as the Field Programmable Gate Array (FPGA) and the Applied Specific Integrated Circuit (ASIC), as well as the development of new programming tools that allow to take complex digital designs to practice in a very small time, now, it is

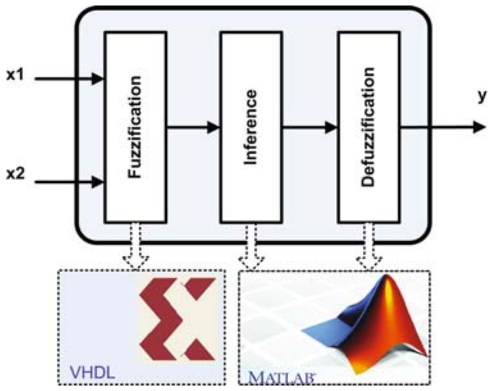


Fig. 1. Overview of the computer architecture used to test and validate the VHDL code of the fuzzification stage.

possible to design and implement high performance systems embedded into an integrated circuit. Nowadays, the study and proposals to implement a fuzzy system into an FPGA is growing up [23].

The main objective of this work is to show how to implement the fuzzification stage of a type-2 fuzzy system based on the average of two type-1 fuzzy systems using the method proposed in [20]. The proposed algorithms were tested using the Xilinx System Generator (XSG) from Xilinx, and the Matlab/Simulink from Mathworks. In [26], the details to implement and validate through the Xilinx System Generator (XSG) the type-1 fuzzification stage were published. Fig. 1 shows the setting up of the software architecture that was used to test and validate the type-1 and type-2 fuzzification stages. Note in this figure, that there is a main block containing three sub-blocks: Fuzzification, inference, and defuzzification. The fuzzification block has two inputs, and one output, this block is related with another block labeled as VHDL (Very High Description Language) to indicate that this stage was programmed using this high level hardware description language; the other two stages (inference and defuzzification) are related with one block labeled as MATLAB to indicate that these stages were programmed using Matlab/Simulink. Regarding the fuzzification stage, two different architectures were proposed, they are: Fuzzification using only the active MFs, and fuzzification using all the MFs. In the first one, only the affected MFs by the inputs were considered. In the second architecture, all the MFs were considered. This work is focused on the design of the fuzzification stage for active MFs.

There are other works related to this one; for example, for type-1 fuzzy systems [13, 15, 18, 19, 22, 24, 25], in the book [17] the authors give an academic example of the VHDL coding of a type-1 fuzzy controller. With respect to type-2 fuzzy systems there are some works like [14, 15, 16].

This work has been organized as follows: Section 2 is devoted to give an introduction from crisp sets to type-1 and type-2 fuzzy sets (FSs), some useful definitions about generalized type-2 fuzzy sets are given with special emphasis in the MFs. In section 3 the mathematics for interval type-2 fuzzy sets is given. In section 4 the average type-2 fuzzy inference system is explained, emphasizing on the fuzzification stage. Section 4.1 was dedicated to explain the most common methods to perform digital fuzzification, we explain in this section the method used in this work, two important piece of VHDL code to fuzzify using triangular and trapezoidal MFs are given; also, we explain the arithmetic method used to handle real and integers numbers to improve the performance, a numeric example is given. Section 5 was dedicated to explain the general idea that was followed in testing and validating this stage. First, using a type-1 FS, an explanation of the input and output MFs, and their respective linguistic variables and terms is given, we show the design entity for this system. Next, it is explained how to implement the type-2 fuzzification stage for the average method, by taking advantage of VHDL code reusability; i.e., using two times the same VHDL code previously developed for the type-1 FS. Moreover, a Simulink model that allows to test the developed type-2 VHDL code for the fuzzification stage is shown. Finally, in section 5.4 some conclusions about this work are given.

2 Sets

This section is dedicated to provide definitions of set theory, starting with "crisp sets", in order to introduce the basic concepts of type-1 and type-2 fuzzy set theory.

2.1 Crisp Sets

In classical set theory (George Cantor, 1845-1918) [2], a set A is comprised of elements x with membership into a universe of discourse X; i.e., $x \in X$. The membership of an element can be expressed using a membership function (MF) that is also known as characteristic function or discrimination function [12]. The definition 1 is valid to represent a MF in the classical bivalent set (crisp set) theory. One of the most common method to represent the crisp set A is,

$$A = \{x | x \in X\} \tag{1}$$

Definition 1. For a crisp set A, the membership function μ_A is defined as

$$\forall x \in X,\ \mu_A(x) = \begin{cases} 1 \text{ if } & x \in A \\ 0 \text{ if } & x \notin A \end{cases} \tag{2}$$

in other words, the function μ_A maps the elements in the universal set X to the set $\{0,1\}$,

$$\mu_A(x) : X \to \{0,1\}. \tag{3}$$

□

2.2 Fuzzy Sets

Ordinary fuzzy sets were developed by Lotfi Zadeh in 1965 [28], they are an extension of classical set theory where the concept of membership was extended to have various grades of membership on the real continuous interval [0,1]. The original idea was to use a fuzzy set (FS); i.e, a linguistic term to model a word; however, after almost ten years, Zadeh introduced the concept of type-n FS as an extension of an ordinary FS (type-1 FS) with the idea of blurring the degrees of membership values [27].

Type-1 fuzzy sets have demonstrated to work efficiently in many applications, most of them use the mathematics of fuzzy sets but losing the focus on words, that are mainly used in the context to represent a function that is more mathematics than linguistic [8]. Membership functions are used to characterize type-1 and type-2 fuzzy sets.

2.2.1 Type-1 Fuzzy Sets

Previously, we mentioned that a FS and its MF is an extension of a crisp set defined in (1) and in (3), respectively. Therefore, that definition has been modified for the FS and the MF. A type-1 FS is a set of ordered pairs expressed by (4) [6], and its corresponding MF is given by definition 2,

$$A = \{(x, \mu_A(x)) | x \in X\} \tag{4}$$

Definition 2. For a type-1 fuzzy set A, each element is mapped to [0,1] by its MF μ_A, where [0,1] means real numbers between 0 and 1, including the values 0 and 1,

$$\mu_A(x) : X \to [0,1]. \tag{5}$$

□

2.2.2 Type-2 Fuzzy Sets

In a type-1 FS the word fuzzy has the connotation of uncertainty, the problem is that once the MF parameters have been specified the type-1 FS is completely certain; as a direct consequence of limitations of type-1 fuzzy sets to handle appropriately linguistic uncertainties type-n fuzzy sets were proposed [28], being type-2 fuzzy sets an special case, the advent of this new type of fuzzy sets came with the fact that uncertainty is always encountered in real life problems and in different ways; so, type-2 fuzzy sets allow to handle appropriately linguistic and random uncertainties [1, 10].

A pointwise definition of a type-2 FS is given as follows,

Definition 3. For a type-2 FS $\widetilde{A}$ characterized by a type-2 MF $\mu_{\widetilde{A}}(x,u)$ were $x \in X$ and $u \in J_x \subseteq [0,1]$, i.e.,

$$\widetilde{A} = \{(x,u),\ \mu_{\widetilde{A}}(x,u) |\ \forall\, x\ \in\ X, \forall\, u\ \in\ J_x\ \subseteq [0,1]\} \tag{6}$$

where $0 \leq \mu_{\widetilde{A}}(x,u) \leq 1$. □

Another way to express $\widetilde{A}$ is,

$$\widetilde{A} = \int_{x\in X}\int_{u\in J_x} \mu_{\widetilde{A}}(x,u)/(x,u) \quad J_x \subseteq [0,1] \tag{7}$$

Similar to type-1 FS, the symbol $\int$ stands for union of continuous functions, and the corresponding symbol for discrete functions is $\sum$; then in (7), $\int\int$ denotes union over all admissible x and u.

Definition 4. The 2D plane with axes u and $\mu_{\widetilde{A}}(x',u)$ formed with $x = x'$ is called a *vertical slice of* $\mu_{\widetilde{A}}(x,u)$. □

Definition 5. A secondary MF is a *vertical slice* of $\mu_{\widetilde{A}}(x,u)$; in particular,$\mu_{\widetilde{A}}(x = x',u)$ for $x' \in X$ y $\forall u \in J_{x'} \subseteq [0,1]$, i.e.,

$$\mu_{\widetilde{A}}(x = x',u) \equiv \mu_{\widetilde{A}}(x') = \int_{u\in J_{x'}} f_{x'}(u)/u \quad J_{x'} \subseteq [0,1]. \tag{8}$$

where $0 \le f_{x'} \le 1$. □

We can drop the prime notation on $\mu_{\widetilde{A}}(x')$ $\forall x' \in X$ and refer to $\mu_{\widetilde{A}}(x)$ as a secondary membership function also known as the secondary set, which is a type-1 FS.

Using the concept of secondary sets, a type-2 FS can be reinterpreted as the union of all the vertical slices,

$$\widetilde{A} = \{(x, \mu_{\widetilde{A}})|\forall x \in X\} \tag{9}$$

or, as

$$\widetilde{A} = \int_{x\in X} \mu_{\widetilde{A}}(x)/x = \int_{x\in X} [\int_{u\in J_x} f_x(u)/u]/x \qquad J_x \subset [0,1]. \tag{10}$$

Definition 6. The *primary membership* of x, J_x ($J_x \subseteq$[0,1] $\forall x \in X$), is the domain of a secondary MF. □

Definition 7. A *secondary grade* is the amplitude of a secondary MF. For example, in (10) $f_x(u)$ is a secondary grade. □

Definition 8. A *main membership function* of a type-2 FS is the union of all the secondary grades equal to 1 (considering that each secondary MF has only one secondary grade equals to 1); i.e.,

$$\mu_{main}(x) = \int_{x\in X} u/x \quad \text{where} \quad f_x(u) = 1 \tag{11}$$

□

Generalized type-2 fuzzy sets are those where secondary membership grades are in the range of [0,1]; but there is the case when secondary grades are always 1, then we have the next definition,

Definition 9. A type-2 FS where its secondary MF is always 1 ($f_x(u) = 1, \ \forall u \in J_x \subseteq [0,1]$), is called an *interval type-2 FS*.

Definition 10. The *footprint of uncertainty* (FOU) is the union of all primary memberships, it is represented as a bounded region that contains uncertainties in the primary memberships of a type-2 FS, $\widetilde{A}$, i.e.,

$$FOU(\widetilde{A}) = \bigcup_{x \in X} J_x \tag{12}$$

□

The concept of FOU is very useful because it allows to focus the attention only on the inherent uncertainties of a type-2 MF. For convenience, the FOU can be described in terms of upper and lower MFs, so we have the next definition.

Definition 11. There are two type-1 membership functions that are the bounds of the FOU, they are the upper and the lower membership functions. The upper bound of $FOU(\widetilde{A})$ is associated with the upper MF, which is denoted by $\bar{\mu}_{\tilde{A}}(x)$, $\forall x \in X$. The lower bound of $FOU(\widetilde{A})$ is associated with the lower MF denoted by $\underline{\mu}_{\tilde{A}}(x)$, $\forall x \in X$, i.e.,

$$\bar{\mu}_{\tilde{A}}(x) = \overline{FOU(\tilde{A})} \quad \forall x \in X. \tag{13}$$

$$\underline{\mu}_{\tilde{A}}(x) = \underline{FOU(\tilde{A})} \quad \forall x \in X. \tag{14}$$

□

Figure 2 shows an interval type-2 MF, the shadow region is the FOU. At the points x_1 and x_2 are the primary MFs J_{x_1} and J_{x_2}, the corresponding secondary MFs $\mu_{\tilde{A}}(x_1)$ and $\mu_{\tilde{A}}(x_2)$ are shown in Figure 3.

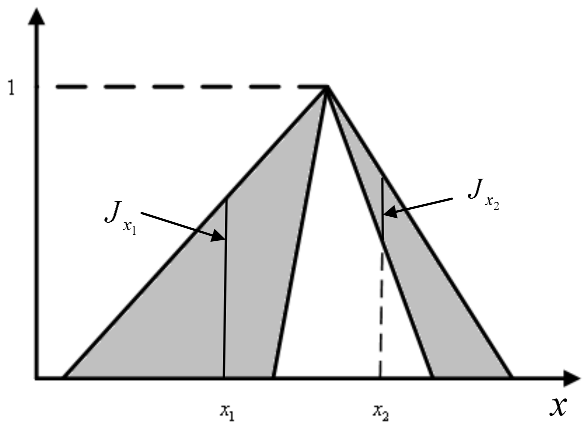


Fig. 2. Type-2 membership function with FOU showing the primary memberships.

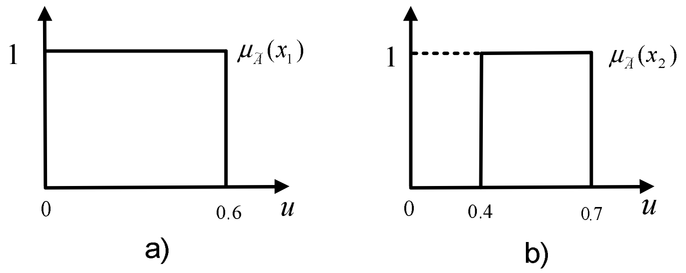


Fig. 3. Secondary membership values a) J_{x_1}, b) J_{x_2}.

3 Interval Type-2 Fuzzification Stage

In section 2.2.2 was mentioned that type-2 FSs were proposed to cope with the uncertainty that might come from several sources; and it was used the term linguistic uncertainties when the FS is used to model a word, this to embrace uncertainties from fuzziness (or vagueness) that result from imprecise boundaries of FSs, non-specificity which is connected with sizes (information based-imprecision), and strife (or discord) that expressed conflicts among the various sets or alternatives [8]. In this section, a summary of type-2 FIS is provided, emphasizing on the fuzzification stage, since it is the aim of this chapter. Figure 4 shows a schematic of a type-2 FIS, it has been divided in four stages: fuzzification, inference engine, type reducer and defuzzification. The task of the fuzzifier is to map a crisp value $x = (x_1, \ldots, x_p)^T \in X_1 \times X_2 \times \ldots \times X_p \equiv \mathbf{X}$ into a type-2 FS $\widetilde{A}_x$ in $\mathbf{X}$. $\widetilde{A}_x$ is a type-2 fuzzy singleton if $\mu_{\tilde{A}_x} = 1/1$ for $x = x'$ and $\mu_{\tilde{A}_x} = 1/0$ for all other $x \neq x'$ [10, 9].

Figure 5 shows an example of the fuzzification process for an interval type-2 FIS, the universe of discourse X is in the range of $[-10, 10]$. Considering that the input x =-4 cuts the type-2 MFs "Negative" and "Zero", for the upper bound of the first MF the membership value is $\bar{\mu}_N$=0.6, and for the lower bound A is $\underline{\mu}_N$=0.5 a set of

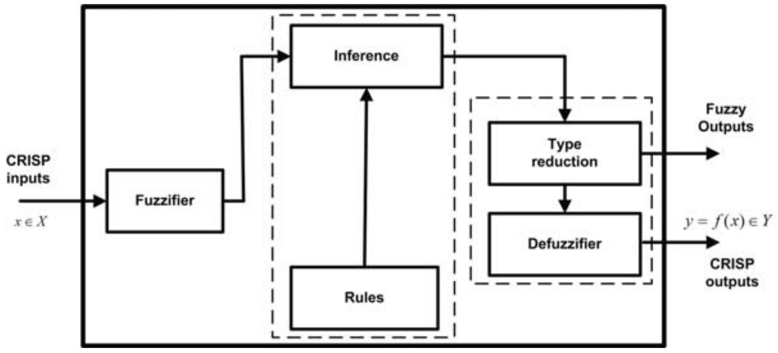


Fig. 4. Type-2 fuzzy system.

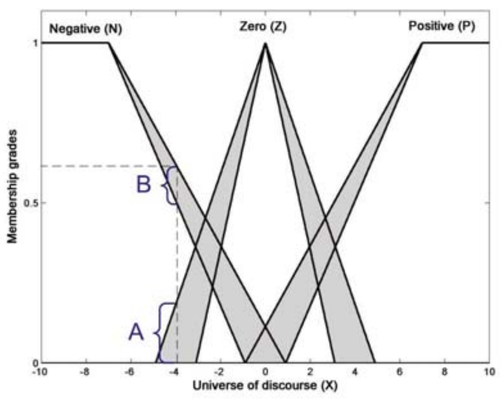


Fig. 5. Interval type-2 MFs.

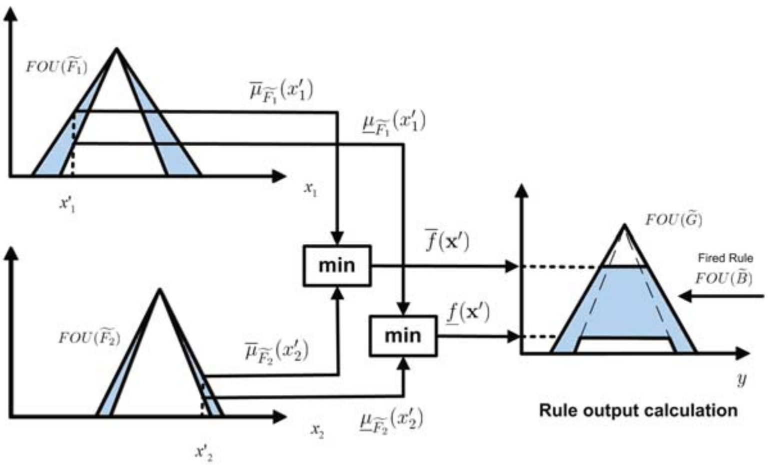

$FOU(\widetilde{F}_1)$
$\overline{\mu}_{\widetilde{F}_1}(x'_1)$
$\underline{\mu}_{\widetilde{F}_1}(x'_1)$
x'_1
x_1
min
$\overline{f}(\mathbf{x}')$
min
$\underline{f}(\mathbf{x}')$
$FOU(\widetilde{F}_2)$
$\overline{\mu}_{\widetilde{F}_2}(x'_2)$
$\underline{\mu}_{\widetilde{F}_2}(x'_2)$
x_2
x'_2
$FOU(\widetilde{G})$
Fired Rule
$FOU(\widetilde{B})$
y
Rule output calculation

Fig. 6. Pictorial description of input and antecedent operations for an interval type-2 singleton, the process uses the minimum t-norm. The average type-2 method is compatible with this idea; however, instead of using type-2 MFs in the antecedents, the method splits each type-2 MF in two type-1 MF placed at the uncertainty boundaries; i.e., $FIS_u(\widetilde{F}_1)$ is used for the $\overline{FOU(\widetilde{F}_1)}$, and $FIS_l(\widetilde{F}_1)$ for $\underline{FOU(\widetilde{F}_1)}$.

primary membership values in the range [0,0.25], the secondary membership values are equal to one for both fuzzy terms. A type-2 fuzzy inference engine combines the if-then rules and gives a mapping from input type-2 fuzzy sets to output type-2 fuzzy sets. Figure 6 shows a pictorial description for the inputs x_1 and x_2, and the minimum t-norm. According to Liang and Mendel [5], for an interval type-2 FIS the result of the input and antecedent operations are contained in the interval type-1 firing set $[\underline{f}^l, \bar{f}^l]$, where

$$\underline{f}^l = \underline{\mu}_{\tilde{F}_1^l}(x_l') \bigstar \cdots \bigstar \underline{\mu}_{\tilde{F}_P^l}(x_p') \tag{15}$$

$$\bar{f}^l = \bar{\mu}_{\tilde{F}_1^l}(x_l') \bigstar \cdots \bigstar \bar{\mu}_{\tilde{F}_P^l}(x_p') \tag{16}$$

Depending on the desired output, it is possible to use the "output processing" block to obtain a type-reduced set (type-1), or a crisp output.

4 Average Type-2 FIS

An average type-2 FIS consists in substituting an interval type-2 FIS (Figure 5) with two type-1 FIS (Figure 7). The main idea of this method is to embrace the uncertainty of the interval type-2 FSs using type-1 FSs. One FIS is used for the upper bound uncertainty, and the second FIS for the lower bound uncertainty. Considering the type-2 FS "A", the interval of uncertainty can be embraced by two type-1 FS. The MFs of the first type-1 FIS are placed such as they match with the upper bound of $\overline{FOU(\tilde{A})}$, i.e., ($\bar{\mu}_{\tilde{A}}(x)$); similarly, the MFs of the second type-1 FIS are chosen to match with the lower bound of $\underline{FOU(\tilde{A})}$, i.e., $\underline{\mu}_{\tilde{A}}(x)$. Each type-1 FIS handles an identical set of rules, however the fuzzy output will be different because the MFs of the inputs are not the same. The average type-2 FIS is consistent with the nomenclature of interval type-2 FIS; so, Fig. 6 remains being useful to explain the fuzzification method that was used in this work. Basically, in this first stage, the difference lies in the fact that an average type-2 FIS uses two type-1 FS in the fuzzification stage to substitute each type-2 FS. A crisp value is obtained by averaging the defuzzified output or the two type-1 FIS.

The first step to split the IT2 FIS is to begin defining that one of the FIS is going to be used for the lower bound of the uncertainty, FIS_l; and the second one for the upper bound, FIS_u. Hence, for the linguistic variables X_1 and X_2 of an IT2 FIS, we will write $FIS_l(X_1)$ for the lower bound of uncertainty, and $FIS_u(X_2)$ for the upper bound. Following a similar idea of nomenclature, the linguistic terms of the IT2 FIS of Figure 6, $FOU(\widetilde{F_1})$ and $FOU(\widetilde{F_2})$, will be written as $FIS_l(\widetilde{F_1},\widetilde{F_2})$, and $FIS_u(\widetilde{F_1},\widetilde{F_2})$ to indicate that the $\underline{FIS_l(\widetilde{F_1},\widetilde{F_2})}$ was developed considering the lower bound of the type-2 linguistic terms $\widetilde{F_1}$ and $\widetilde{F_2}$; similarly, $FIS_u(\widetilde{F_1},\widetilde{F_2})$ indicates the use of the upper bounds of uncertainty of the mentioned terms.

4.1 Type-2 Fuzzification Stage Architecure

In Figure 1 the proposal to test and validate the type-2 fuzzification stage coded in VHDL was illustrated. This figure depicts that the inference and defuzzification stages were programmed in Matalb/Simulink using the existing Fuzzy Simulink modules of the Fuzzy toolbox, whereas the fuzzification stage was coded in VHDL, this was achieved to evaluate only the VHDL codification.

As was mentioned before, the type-2 FIS explained in this work is based on calculating the average of two type-1 FIS (for the upper and lower bounds).

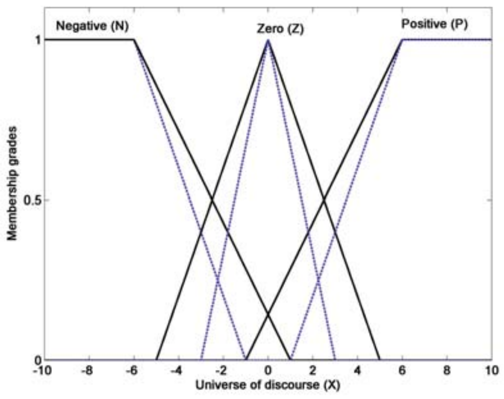


Fig. 7. The interval type-2 MFs of Fig. 5 were substituted with type-1 MFs. The MFs plotted with solid lines correspond to the fuzzification stage of one FIS, so we have for the upper bounds of uncertainty $FIS_u(\widetilde{N},\widetilde{Z},\widetilde{P})$; and for the lower bounds, $FIS_l(\widetilde{N},\widetilde{Z},\widetilde{P})$.

Microelectronic implementations of the fuzzification stage can be achieved in two ways: Storage memory and arithmetic calculation. Next, a brief review of each method is given.

4.1.1 Storage Memory Method

This method consists in sampling the universe of discourse and saving into memory the corresponding membership values and linguistic terms. This alternative allow us to define any shape of membership function, since the only requirement is to sample and save the membership values that are addressed by their corresponding binary input in the universe of discourse. One limitation of this method is regarding precision, because it depends on the number of samples and bits needed for an application, high precision applications require higher memory resources. In [25] can be found more information about the amount of memory required to achieve this implementation.

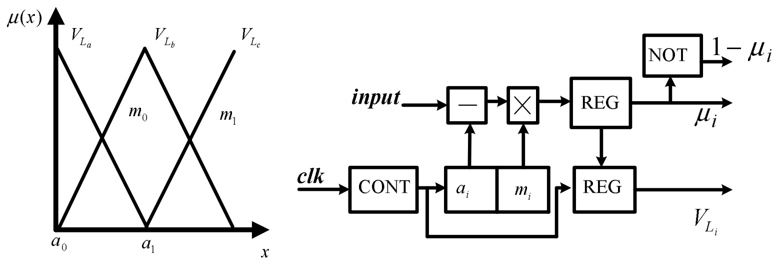


Fig. 8. Block diagram to illustrate the arithmetic calculation method.

4.1.2 Arithmetic Calculation Method

The arithmetic method consists in performing a progressive calculation of the degree of membership of the antecedents. This method is restricted to use only symmetric triangular MFs. The method uses two memory allocations for each straight line, where their slopes values m_i, and interception points a_i are saved into memory. An arithmetic circuit for each input shown in Fig. 8(a), solves the corresponding straight-line equation to obtain the membership value, as can be seen in Fig. 8(b). A counter driven by the system clock generates the memory addresses, the breakpoint value is subtracted from the input and the result is multiplied by the slope; if $x > xi$, the product is stores in a latch an the counter is increased in the next clock pulse. The process continues until $x < xi$. This circuit requires as many clock cycles to perform its operation as fuzzy sets are defined.

4.1.3 Novel Arithmetic Fuzzification Method

The arithmetic fuzzification method (AFM) was originally proposed by the authors in [26], in this work some improvements to the original proposal are presented. The AFM method is more versatile than the storage memory and the arithmetic calculation methods that were previously explained.

In comparison with the storage memory method explained in section 4.1.1, this method does not need to have a preloaded singleton table values nor to use high memory resources to obtain high resolution values.

Comparing with the arithmetic calculation method of section 4.1.2, the AFM allows to use symmetric and non-symmetric triangular and trapezoidal MFs. The intersection and slopes values can be calculated online and updated, which is a nice capability to implement adaptive fuzzy systems. In addition, the AFM only needs one clock cycle to achieve the fuzzification, this is a very attractive feature when

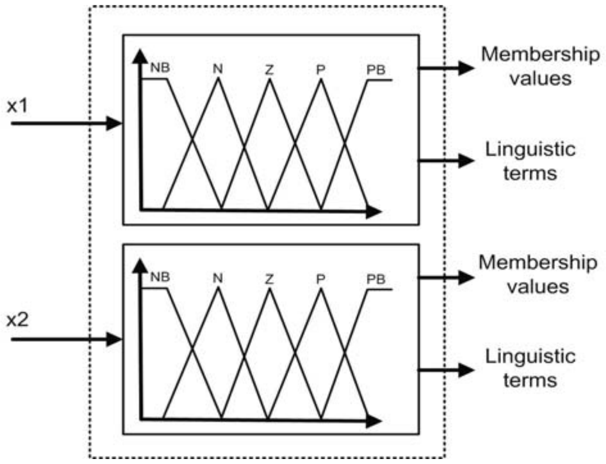


Fig. 9. Fuzzification stage with two inputs. For each input there are two outputs: The membership value, and the linguistic term.

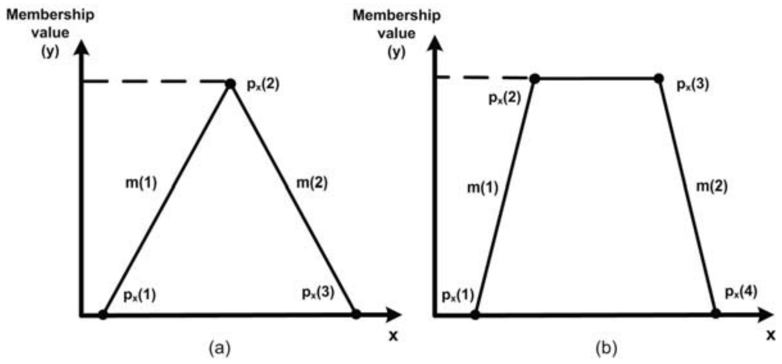


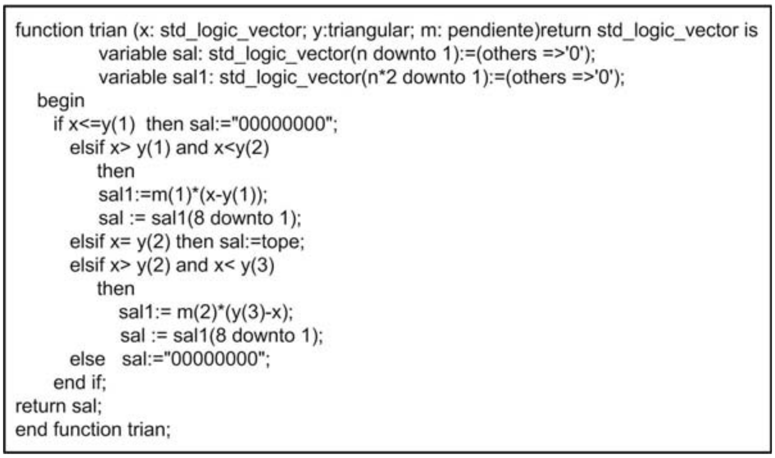
Fig. 10. Triangular and trapezoidal MFs. The slopes and intercept points are shown.

we compare with the arithmetic calculation method of section 4.1.2 that requires several clock cycles to perform this task.

The AFM code is divided in two parts. The first part is parallel code devoted to update the intersection points, calculate slopes values, and assign the corresponding linguistic tag to each MF. It uses one status bit (status flag) related with each slope to inform that the corresponding slope value has been updated. Once the calculations have been achieved, all the status flags are set. If any change occurs, the implied status flags should be reset. The second part of the code checks, whether all the status flags are set; if so, then the crisp input is fuzzyfied. Fig. 9 is a big picture of how the fuzzification is achieved for a two input system, the outputs are the membership grades and linguistic terms of the active MFs. The AFM for a given input (crisp value) can be summarized into three steps:

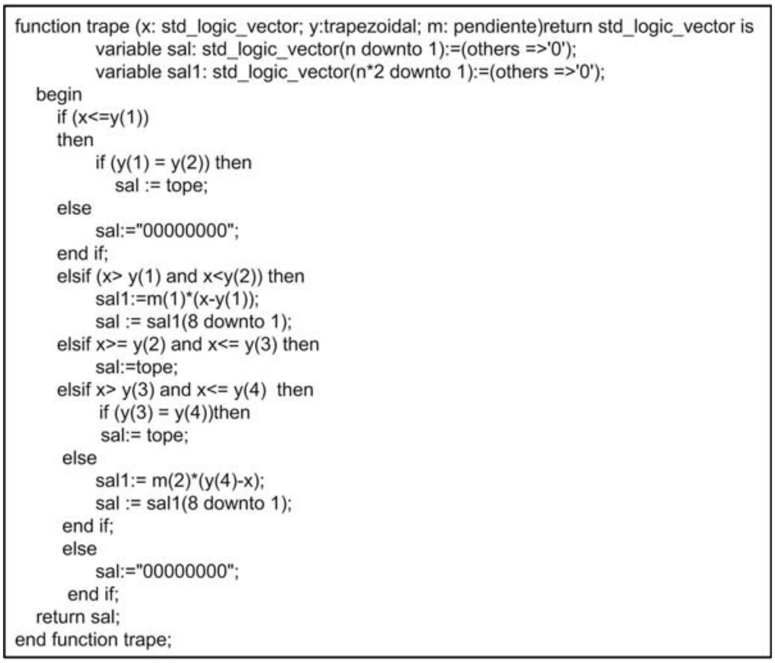
1. Calculate the slope values.
2. Calculate the membership values.
3. Assign to signals the membership values and linguistic terms.

Figures 10(a) and 10(b) show the slopes and intercepts of the straight lines of a triangular MF, and a trapezoidal MF, respectively. Both, use the basic straight line equation given by the slope-intercept form $y = mx + b$ where "m" is the slope and 'b' is the intercept. The MFs have two slopes, the positive slope "$m(1)$", and the negative slope "$m(2)$"; although they have different signs, both are handled as positive to avoid signed operations; later, the sign is handled by the software. Figure 11 and 12 show the VHDL code to implement the triangular and trapezoidal MF, respectively. These MFs can be implemented using fixed point arithmetic to represent integers and real numbers, as well as floating point arithmetic. The decision of the best numeric representation most of the time depends on how the problem will be handled in later stages. This decision must be done carefully at an early project phase, in a broad sense, fixed point implementations provide higher speeds and lower cost implementations, on the other hand, floating point offers higher dynamic range and it is not necessary to perform scaling task that is a very attractive feature for complicated algorithms. Its main virtue is that it allows easy conversion to



```
function trian (x: std_logic_vector; y:triangular; m: pendiente)return std_logic_vector is
            variable sal: std_logic_vector(n downto 1):=(others =>'0');
            variable sal1: std_logic_vector(n*2 downto 1):=(others =>'0');
   begin
     if x<=y(1)  then sal:="00000000";
       elsif x> y(1) and x<y(2)
            then
            sal1:=m(1)*(x-y(1));
            sal := sal1(8 downto 1);
       elsif x= y(2) then sal:=tope;
       elsif x> y(2) and x< y(3)
            then
               sal1:= m(2)*(y(3)-x);
               sal := sal1(8 downto 1);
       else   sal:="00000000";
     end if;
return sal;
end function trian;
```

Fig. 11. VHDL codification to implement the triangular MF.



```
function trape (x: std_logic_vector; y:trapezoidal; m: pendiente)return std_logic_vector is
            variable sal: std_logic_vector(n downto 1):=(others =>'0');
            variable sal1: std_logic_vector(n*2 downto 1):=(others =>'0');
   begin
      if (x<=y(1))
      then
            if (y(1) = y(2)) then
               sal := tope;
      else
            sal:="00000000";
      end if;
      elsif (x> y(1) and x<y(2)) then
            sal1:=m(1)*(x-y(1));
            sal := sal1(8 downto 1);
      elsif x>= y(2) and x<= y(3) then
            sal:=tope;
      elsif x> y(3) and x<= y(4)  then
            if (y(3) = y(4))then
            sal:= tope;
       else
            sal1:= m(2)*(y(4)-x);
            sal := sal1(8 downto 1);
       end if;
       else
            sal:="00000000";
        end if;
   return sal;
end function trape;
```

Fig. 12. VHDL codification to implement the trapezoidal MF.

decimal digits for printing or display and faster decimal calculations, its drawbacks are the increased complexity of circuits needed to implement mathematical operations and a relatively inefficient encoding that occupies more space than a pure binary representation.

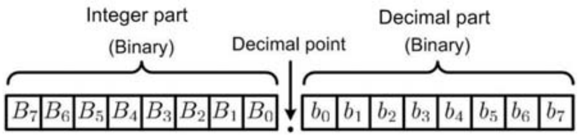

Integer part
(Binary)
Decimal point
Decimal part
(Binary)
B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7

Fig. 13. VHDL code for the triangular MF.

4.1.4 Fuzzification Using Real Numbers

The arithmetic used in this work is based on the encoding method called Decimal Floating Point Binary Encoding (DFPBE), but differently to the common way to implement this encoding method, we used a fixed point format with no exponents. Figure 13 shows the general idea of encoding a real number using two integers, and to calculate the slope "$m(1)$" of the triangular MF showed in Figure 10(a), it is necessary to achieve a method with several steps; so, Fig. 16 is going to be used in order to explain the algorithm. Considering that we are going to fuzzify the crisp binary value 48 (i.e, x1=48), to obtain $\mu(48) = 0.3325$. First, it is necessary to calculate the positive slope of the Negative (N) MF, using the slope formula $m = \frac{y}{x}$, i.e., $m = \frac{255-0}{80-32} = 5.3125 \approx 5.3$ since the binary equivalent for an eight-bits MF is 255, hence the membership value is $\frac{5.3\times(48-32)}{255}$.

1. Obtain the divisor of the slope formula: Considering that the x value is given by the points $p_x(1)$=32 and $p_x(2)$=80, the divider is obtained using $p_x(2) - p_x(1) = 80 - 32 = 48$. Since the codification method DFPBE is being used, it is necessary for convenience to divide by ten the divider, instead of multiplying by ten the dividend as is common; therefore, we have $\frac{p_x(2)-p_x(1)}{10} = 4.8$, then the divider for the integer numbers in the range of (0 to 255) is 48; $Quotient_1(4)$, $residue_1(8)$; hence, then the divider for the decimal part is 4.
2. Obtain the slope: The slope is given as the number "$B_m.b_m$"; where "B_m" is an integer number, and "b_m" is a decimal number.
For the integer part: $Bm = \frac{255}{p_x(2)-p_x(1)} = 5.3$: $Quotient_2(5), residue_2(15)$. For the decimal part, $bm = \frac{residue_2}{Quotient_1} = \frac{15}{4} = 3.7 : Quotient_3(3), residue_3(3)$. The slope is a two bytes binary numbers B_m=5, b_m=3; i.e. slope "m(1)=5.3".
3. The membership value for a given crisp input, for example "x=48", is obtained as follows: Adjust the crisp input to the corresponding slope, in this case calculate $x_{48} = x - p_x(1) = 48 - 32 = 16$. Using the slope formula $y = mx$, which is equivalent to $y = B_m x + \frac{b_m}{10} x$. This applied to our example considering scaling, $y = \frac{5(16)+0.3(16)}{255} = 0.3325$; so, the membership value is 0.3325 for a crisp input "x=48".

Note that DFPBE method was used only to calculate the slopes, the membership values are real numbers coded in one binary word.

5 Experiment Design and Validation

In Fig. 1 the general idea followed in testing and validating the fuzzification stage for type-1 and type-2 FIS was presented. More specifically, the FIS was developed to work as a PD (Proportional Derivative) controller into a PD+I (PD+ Incremental) controller configuration to regulate the speed of a DC motor. The next two sections of this chapter are dedicated to explain how to achieve the codification of the type-1 and type-2 fuzzification stages.

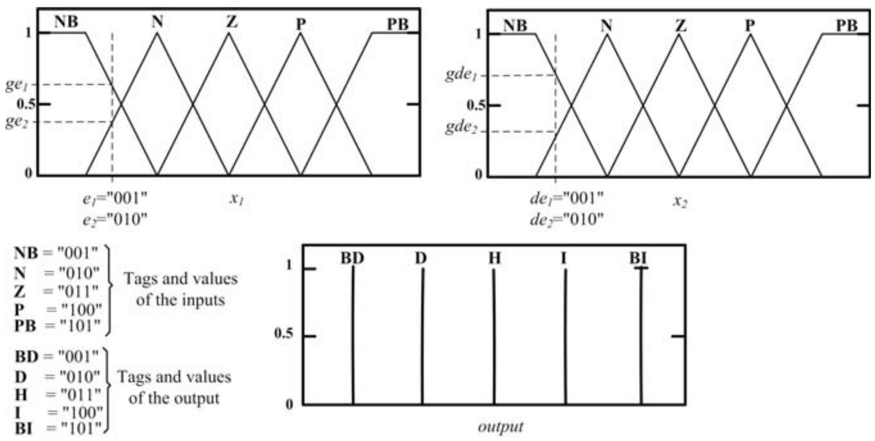


Fig. 14. Fuzzy inputs and outputs. Each fuzzy input has five MFs called Negative Big (NB), Negative (N), Zero (Z), Positive (P), Positive Big (PB). The output has five singletons called Big Decrease (BD), Decrease (D), Hold (H), Increase (I), Big Increase (BI).

5.1 Type-1 Fuzzification Stage: Active MFs

Fig 14 shows the linguistic variables for the inputs and for the output, these are: *error*, *change of error*, and *output*. Each input has five MFs, Negative Big (**NB**), Negative (**N**), Zero (**Z**), Positive (**P**), and Positive Big (**PB**); and each label an associated binary value, for example **NB** is associated with the value **"001"**, etc. The output has five singletons values called Big Decrease (**BD**), Decrease (**D**), Hold (**H**), Increase (**I**), and Big Increase (**BI**); hence, **BD** is associated with the binary value **"001"**, etc. [19, 18]. Additionally, it is shown that each input produces two fuzzified values; i.e., in input 1 (x_1) the dashed line indicates a crisp value that is being fuzzified, so it will produce the variables e_1 and e_2, associated with their corresponding degree of membership stored in variables ge_1 and ge_2. The variable content value for e_1 is "010", and "001" for e_2. Input x_2 works in a similar way. Figure 15 shows the obtained design entity for the aforementioned design, at the output are the active grades values and their corresponding linguistic terms. The variable values for e_1, e_2, ge_1, and ge_2 are assigned to output signals. After debugging, only some minor modifications have to be performed to the VHDL code to take the tested code to the

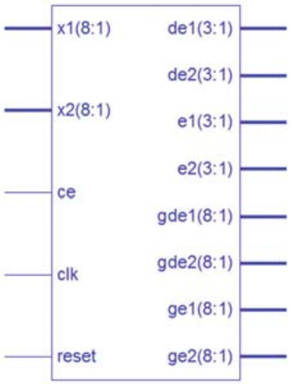


Fig. 15. Design entity of the fuzzification stage. The "clk" input is 50 Mhz for the Spartan 3 (XC3S1000). The "ce" (clock enable input) enables the algorithm in the Simulink. The "reset" input restart the variables of the algorithm.

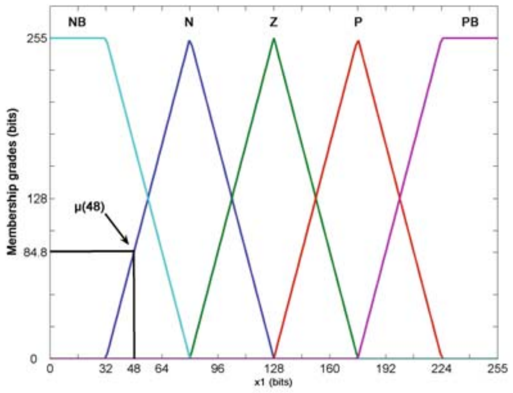


Fig. 16. MFS for the input x_1. An input x1=48 is fuzzified; i.e., $\mu(48) = 84.8$, because we are using an 8-bits binary codification system, rounding towards towards minus infinity, $\mu(48) \approx 84$.

physical implementation into the FPGA. In Fig. 16 we have a crisp input x_1=48 that is fuzzified using type-1 MFs.

5.2 Type-2 Fuzzification Stage: Active MFs

The implementation of the type-2 fuzzification stage for the average method [20, 21] is achieved using two type-1 fuzzification stages; in other words, two instances like the one shown in Figure 15 are used, the only difference between them is the parameters values of the MFs, with the purpose to represent the existing FOU in an interval type-2 MF, the rules and defuzzification stage are similar, although each

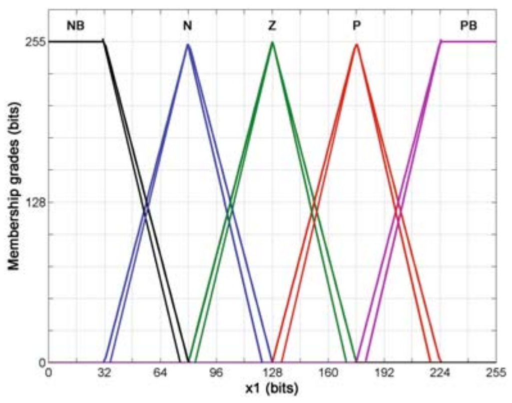


Fig. 17. MFs for the input x_1 of the type-2 fuzzification stage.

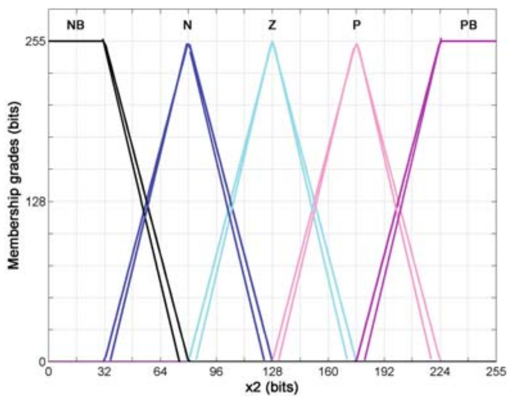


Fig. 18. MFs for the input x_2 of the type-2 fuzzification stage.

system performs independently, and only after each FIS has calculated its respective output the averaging operation is achieved. Figures 17 and 18 show the type-2 MFs used for the "error" and "change_of_error" inputs, each variable has five linguistic terms; however, to take this approach to practice we will need ten MFs for each linguistic variable (two MFs for each linguistic term); hence two FIS are used, the $FIS_u(X_1,X_2)$ for the $\overline{FOU}$ of X_1 ("error" input) and X_2 ("change of error" input); and a second FIS, the $FIS_l(X_1,X_2)$ for the $\underline{FOU}$ of X_1 and X_2. Note, that the signal "error" is connected to X_1, and the signal "change of error" to X_2 of the fuzzification entity in Fig. 15.

5.3 Simulink Model for the Type-2 Fuzzification Stage

Fig. 19 shows the Simulink model that was used to test the type-2 FIS. The type-1 FIS identified by the circled numbers 1, 2, and 3 represent the upper bound FOU;

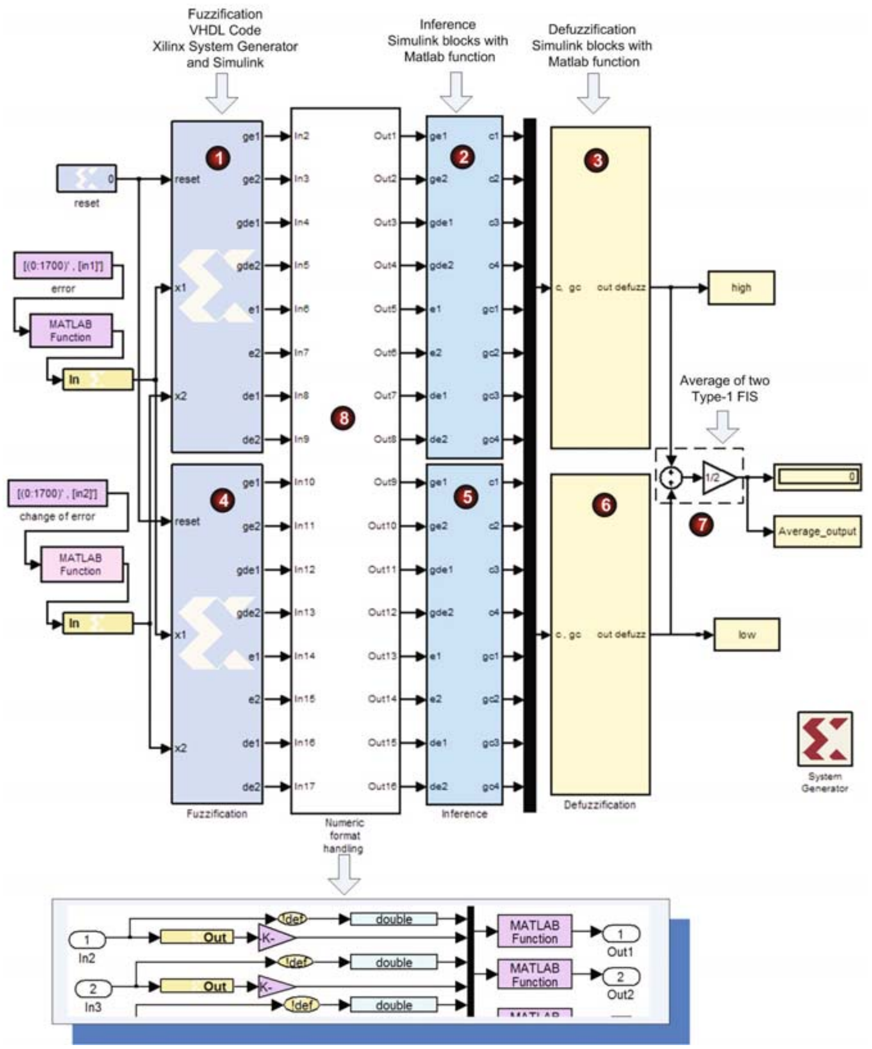

Fuzzification VHDL Code Xilinx System Generator and Simulink
Inference Simulink blocks with Matlab function
Defuzzification Simulink blocks with Matlab function
Average of two Type-1 FIS
reset
error
change of error
MATLAB Function
Fuzzification
Numeric format handling
Inference
Defuzzification
high
low
Average_output
System Generator

Fig. 19. Simulink model of the type-2 FIS (Average method) used to plot the control surface.

and the numbers 4, 5, and 6 identify the second type-1 FIS for the lower bound FOU. The average of the output of both FIS is achieved in the section marked with number 7. In the blocks 1 and 4 are the fuzzification stages, these blocks belongs to the XSG, they are used to import the VHDL code to Simulink. The inference engine of each type-1 FIS are in the blocks 2 and 5. The defuzzification stage of each type-1 FIS are identify by the numbers 3 and 6. Blocks 2, 3, 5, and 6 were programmed using Simulink blocks from the toolbox as well as Matlab functions, they were not programmed in VHDL because in this part the goal is only to test fuzzification stage. The subsystem block number 8 is to handling numeric format between the code of

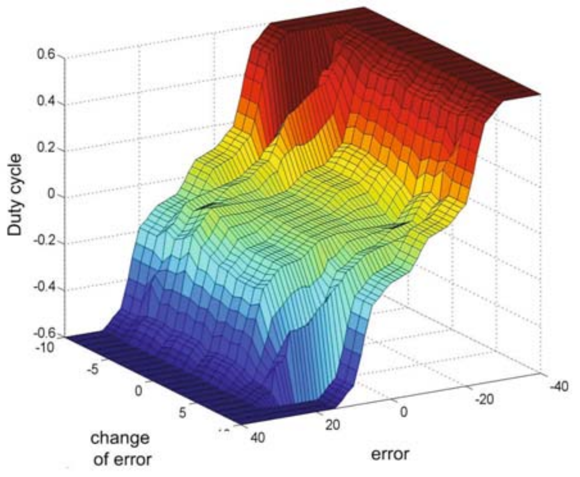


Fig. 20. Control surface obtained with the Simulink model (Average method) showed in Fig. 19.

the VHDL fuzzification stages and the Matlab inference engines, below this block there is a shadowed box showing a small section of the content of this subsystem, the rest is quite similar. The other small Simulink blocks that are at the input of both fuzzification blocks give two vectors of 1700 values each, for the "error" and "change of error" inputs in order to obtain the plot shown in Fig. 20, the small Simulink blocks that are shown between the the fuzzification and inference engine stages are used to achieve numeric compatibility from the numeric codification of real numbers used in the fuzzification stages coded in VHDL, to the floating point numeric representation used in Matlab.

5.4 Conclusions

One of the most stressing task in the development of new methods to improve a system is related with the use of new software and/or hardware technologies. This is specially true when the target of the developed software is code for a VLSI circuit, for example an FPGA. The use of type-2 FIS for real world applications has been plenty demonstrated in many works; so, the idea of taking some existing fuzzy controllers from type-1 to type-2 technology is very appealing. However, we have to expect that the programming cost in terms of developing time grew, as well as the size of code, and performance, this depending on the selected algorithm. There are several proposals to implement an interval type-2 FIS [3, 4, 5], but there is no way to go with little effort from interval type-1 to interval type-2 FIS. The average method provides an option to handle uncertainty similarly than an interval type-2 FIS does; moreover, the method makes easier the transition from type-1 to type-2 FIS, basically the only additional thing that designers will need do is to duplicate and modify the MFs values of the original type-1 FIS. This advantage is more emphatic when the type-1 FIS was coded using VHDL, because this language uses the concept of design entity that can contain a whole design that can be duplicated as many times as needed. Hence, the cost of programming time is reduced to the minimal, the code is practically twice the size of the type-1 FIS, and the performance of the hardware implementation of a type-2 is almost the same than the type-1 FIS, the small difference is due to the time that the electronics takes to perform an averaging operation, i.e; one addition and one divide operation. The Xilinx System Generator takes VHDL code to the Simulink environment offering advantages, since it is possible to analyze the system's behavior using the control surface, or achieving the control of the physical plant by interfacing the Simulink model with real world.

Acknowledgements. The authors thank Comisión de Operación y Fomento de Actividades Académicas del Instituto Politécnico Nacional, Instituto Tecnológico de Tijuana, and CONACYT for supporting our research activities.

References

1. Klir, G.J., Wierman, M.J.: Uncertainty-Based Information: Elements of Generalized Information Theory. Physica-Verlag (1999)
2. Cantor, G.: Contributions to the Founding of the Theory of Transfinite Numbers. Dover Publications Inc. (1915)
3. Wu, H., Mendel, J.M.: Uncertanty Bounds and Their use in the Design of Interval type-2 Fuzzy Logic Systems. IEEE Transactions on Fuzzy Systems 10, 1–16 (2002)
4. Karnik, N.N., Mendel, J.M., Liang, Q.: Type-2 Fuzzy Logic Systems. IEEE Transactions on Fuzzy Systems 7, 643–658 (1999)
5. Liang, Q., Mendel, J.M.: Interval Type-2 Fuzzy Logic Systems: Theory and Design. IEEE Trans. on Fuzzy Systems 8, 535–550 (2000)
6. Jang, J.S.R., Sun, C.T., Mizutani, E.: Neuro-Fuzzy and Soft Computing. Prentice-Hall, Englewood Cliffs (1997)
7. Mendel, J.M.: Type-2 Fuzzy Sets and Systems: an Overview. IEEE Computational Intelligence Magazine 2, 20–29 (2007)
8. Mendel, J.M.: Type-2 Fuzzy Sets: Some Questions and Answers. IEEE Neural Networks Society, United States (2003)
9. Mendel, J.M., John, R.I.B.: Type-2 Fuzzy Sets Made Simple. IEEE Transactions on Fuzzy Systems 10, 117–127 (2002)
10. Mendel, J.M.: Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions. Prentice-Hall, Upper-Saddle River (2001)
11. Karnik, N., Mendel, J.M.: Centroid of a Type-2 Fuzzy Set. Information Sciences 132, 195–220 (2001)
12. Lee, K.H.: Advances in Soft Computing: First Course on Fuzzy Theory and Applications. Springer, Germany (2005)
13. Lago, E., Jiménez, C.J., López, D.R., Sánchez-Solano, S., Barriga, A.: XFVHDL: A Tool for the Synthesis of Fuzzy Logic Controllers. Design Automation and Test in Europe, 102–107 (1998)
14. Melgarejo, M., Peña-Reyes, C.A.: Implementing Interval type-2 Fuzzy Processors. IEEE Computational Intelligence Magazine 2, 63–71 (2007)
15. Miguel, A., Melgarejo, R., Peña-Reyes, C.A.: Hardware architecture and FPGA implementation of a type-2 fuzzy system. In: Proceedings of the 14th ACM Great Lakes symposium on VLSI, pp. 458–461 (2004)

16. Melgarejo, M.A., Garcia, R.A., Peña-Reyes, C.A.: Pro-Two: a hardware based platform for real time type-2 fuzzy inference. In: Proceedings of 2004 IEEE International Conference on Publication Fuzzy Systems, vol. 2, pp. 977–982 (2004)
17. Cirstea, M.N., Dinu, A., Khor, J.G., McCormick, M.: Neural and Fuzzy Logic Control of Drives and Power System, Newnes (2002)
18. Montiel, O., Maldonado, Y., Sepúlveda, R., Castillo, O.: Simple tuned fuzzy controller embedded into an FPGA. In: 2008 NAFIPS Conference Proceedings, pp. 1–6 (2008)
19. Montiel, O., Olivas, J., Sepúlveda, R., Castillo, O.: Development of an Embedded Simple Tuned Fuzzy Controller. In: 2008 IEEE World Congress on Computational Intelligence (WCCI 2008), pp. 555–561 (2008)
20. Sepúlveda, R., Castillo, O., Melín, P., Díaz, A.R., Montiel, O.: Experimental Study of Intelligent Controllers under Uncertainty using type-1 and type-2 Fuzzy Logic. Information Sciences 177, 2023–2048 (2007)
21. Sepúlveda, R., Castillo, O., Melín, P., Montiel, O.: An Efficient Computational Method to implement Type-2 Fuzzy Logic in Control Applications. In: Analysis and Design of Intelligent Systems Using Soft Computing Techniques. Advances in soft computing, vol. 41, pp. 45–52. Springer, Heidelberg (2007)
22. Vuong, P.T., Madni, A.M., Voung, J.B.: VHDL Implementation for a Fuzzy Logic Controller. In: World Automation Congress, WAC apos;2006, pp. 1–8 (2006)
23. Iregui, S., Linares, D., Melgarejo, M.: Performance Evaluation of Fuzzy Operators for FPGA Technology. In: NAFIPS 2008, New York (2008)
24. Sánchez-Solano, S., Senhadji, R., Cabrera, A., Baturone, I., Jiménez, C.J., Barriga, A.: Prototyping of Fuzzy Logic Based Controllers Using Standard FPGA Development Boards. In: 13th IEEE International Workshop on Rapid System Prototyping on Volume, pp. 25–32 (2002)
25. Sánchez Solano, S., Barriga, A., Jiménez, C.J., Huertas, J.L.: Design and Application of Digital Fuzzy Controllers. In: Sixth IEEE International Conference on Fuzzy Systems (FUZZ-IEED 1997), Barcelona, Spain, vol. 2, pp. 869–874 (1997)
26. Maldonado, Y., Montiel, O., Sepuúlveda, R., Castillo, O.: Design and simulation of the fuzzification stage through the Xilinx System Generator. In: Soft Computing for Hybrid Intelligent Systems, vol. 154, pp. 297–305. Springer, Heidelberg (2008)
27. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning. Information Sciences 8, 199–249 (1975)
28. Zadeh, L.A.: Fuzzy sets. Information and Control 8, 338–353 (1965)