New Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics Volume II: Applications

Editors

Krassimir T. Atanassov
Władysław Homenda
Olgierd Hryniewicz
Janusz Kacprzyk
Maciej Krawczak
Zbigniew Nahorski
Eulalia Szmidt
Sławomir Zadrożny

SRI PAS  **IBS PAN**

New Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics Volume II: Applications

New Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics
Volume II: Applications

Systems Research Institute
Polish Academy of Sciences

New Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics
Volume II: Applications

Editors

Krassimir T. Atanassov
Władysław Homenda
Olgierd Hryniewicz
Janusz Kacprzyk
Maciej Krawczak
Zbigniew Nahorski
Eulalia Szmidt
Sławomir Zadrożny

IBS PAN  **SRI PAS**

Systems Research Institute
Polish Academy of Sciences
Newelska 6, 01-447 Warsaw, Poland
www.ibspan.waw.pl

ISBN 83-894-7541-3

Dedicated to Professor Beloslav Riečan on his 75th anniversary

Reconfigurable hardware for fuzzy controller

Paulo Renato de Souza e Silva Sandres, Nadia Nedjah and Luiza de Macedo Mourelle

Department of Electronics Engineering and Telecommunications,
Engineering Faculty,(State University of Rio de Janeiro),
Rua São Francisco Xavier, 524, Sala 5022-D, Maracanã,
Rio de Janeiro, Brazil
psandres@globo.com, nadia@eng.uerj.br, ldmm@eng.uerj.br

Abstract

Computational system modeling is full of ambiguous situations, wherein the designer cannot decide, with precision, what should be the outcome of the system. In [29], L. Zadeh introduced for the first time the concept of *fuzziness* as opposed to *crispiness* in data sets. When he invented *fuzzy sets* together with the underlying theory, Zadeh's main concern was to reduce system complexity and provide designer with a new computing paradigm that allow the to approximate results. Whenever there is uncertainty, *fuzzy logic* together with *approximate reasoning* apply. Fuzzy logic and approximate reasoning [30, 31] can be used in system modeling and control as well as data clustering and prediction [23], to name only few appropriate applications. Furthermore, they can be applied to any discipline such as finance [6], image processing [35, 9], temperature and pressure control [34, 14], robot control [11, 24], among many others.

Process control is one of the many applications that took advantage of the fuzzy logic. Controller are usually embedded into the controller device. This chapter aims at presenting the development of a reconfigurable efficient architecture for fuzzy controllers, suitable for embedding. The architecture is parameterizable so to allows the setup and configuration of the controller so it can be used for various problem applications.

This chapter describes the hardware implementation of a shell fuzzy controller based on generic fuzzy logic, allowing the setup and configuration for various problem situations.

New Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics. Volume II: Applications *(K.T. Atanassow, W. Homenda, O. Hryniewicz, J. Kacprzyk, M. Krawczak, Z. Nahorski, E. Szmidt, S. Zadrożny, Eds.), IBS PAN - SRI PAS, Warsaw, 2012.*

1 Introduction

The Fuzzy Logic is a subject of great interest in scientific circles, but it is still not commonly used in industry, as it should be. Eventually, we found some literature containing practical applications that is being currently used in industry [16, 22]

Fuzzy logic has been used in many of applications, such as expert systems, computing with words, approximate reasoning, natural language, process control, robotics, modeling partially open systems, pattern recognition, decision making and data clustering [20] and [26].

There are many related works that implemented a fuzzy controller on a FPGA, but most of them present controller designs that are only suitable for a specific application. Mainly, the designs do not use 32-bit floating-point data [18] [21] [25] [17] [5]. The floating-point data representation is crucial for the sensibility of the controller design. In contrast, all the required computation in the proposed controller are performed by a simple precision floating-point co-processor.

The purpose of the development of a reconfigurable hardware of a *shell* fuzzy controller, that can include any number of inputs and outputs as well as any number of rules, is the possibility of creating a device that can be used more widely and perhaps spread the concept of fuzzy logic in the industrial final products.

This chapter is divided into three sections. First, in Section 2, we introduce briefly some concepts of fuzzy controller, which will be useful to follow the description of the proposed architecture. Then, in Section 3, we describe thoroughly, the macro-architecture of the fuzzy controller developed. After that, in Section 4, we give details about the main components included in the macro-architecture. Subsequently, in Section 5, we show, via simulation snapshots, that the proposed architecture is functionally operational. Finally, in Section 6, we draw some conclusions and point out some new direction for the work in progress.

2 Fuzzy Controllers

Fuzzy control, which directly uses fuzzy rules, is the most important and common application of the fuzzy theory [27]. Using a procedure originated by E. Mamdani [15], three steps are followed to design a fuzzy controlled machine:

1. fuzzification or encoding: This step in the fuzzy controller is responsible of encoding the crisp measured values of the system parameter into a fuzzy term using the respective membership functions;
2. inference: This step consists of identifying the subset of fuzzy rules that can be fired, i.e. those with antecedent propositions with truth degree not zero, and draw the adequate fuzzy conclusions;

3. defuzzification or decoding: This is the reverse process of fuzzification. It is responsible of decoding a fuzzy variable and compute its crisp value.

The generic architecture of a fuzzy controller is given in Fig. 1. Its main components consist of a *knowledge repository*, the *encoder* or *fuzzification* unit, the *decoder* or *defuzzification* unit and the *inference engine*. The knowledge base stores two kind of data: the fuzzy rules which are required by the inference engine to reach the expected results and knowledge about the fuzzy terms together with their respective membership functions as well as information about the universe of discourse of each fuzzy variable manipulated within the controller. The encoder implements the transformation from crisp to fuzzy and the decoder the transformation from fuzzy to crisp. Of course, the inference engine is the main component of the controller architecture. It implements the approximate reasoning process.

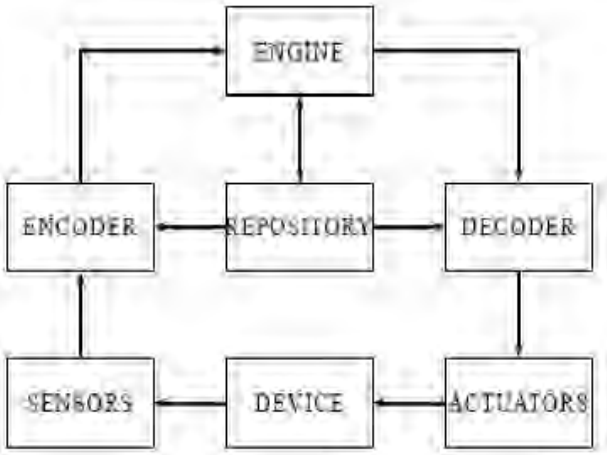


Figure 1: Generic architecture of fuzzy controllers

2.1 Operation

In order to explain thoroughly how a fuzzy controller operates, we borrow the famous example of a the inverted pendulum from [3]. The inverted pendulum is described in Fig. 2. The problem consists of controlling the movement of a pole on a mobile platform. It can only balance to the right or left.

The linguistic variables are the *angle* between the platform and the pendulum ([-30°, 30°]), the angular *velocity* of this angle ([-15,15]) and the *speed* of the platform ([-3, 3]). The first two variables are the controller input while the third one is the expected controller output. The fuzzy terms for each of the identified

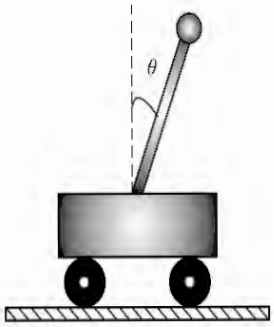


Figure 2: Inverted Pendulum

variables are given in Table 1. The membership functions of the controller linguistic fuzzy terms *angle*, *velocity* and *speed* are given in Fig. 3, Fig. 4 and Fig. 5 respectively.

Table 1: Fuzzy variable and corresponding linguistic terms

angle – A	*velocity – V*	*speed – S*
n-large	n-high	n-fast
n-small	n-low	n-slow
insignificant	null	still
p-small	p-low	p-slow
p-large	p-high	p-fast

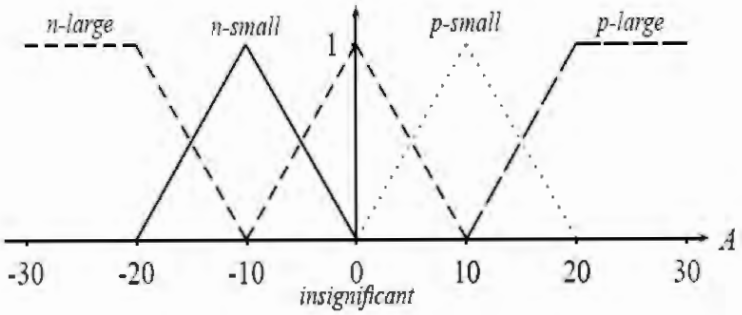


Figure 3: Fuzzy representation of membership function μ_{angle}

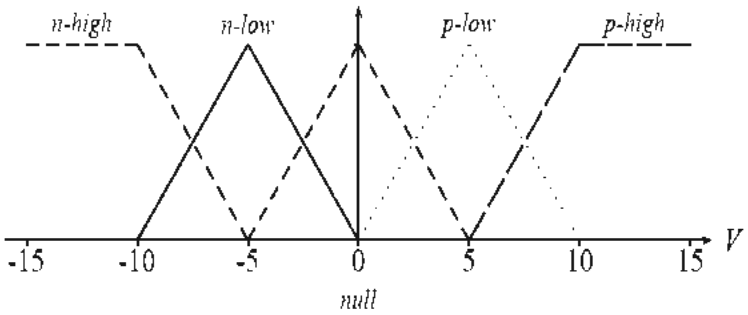


Figure 4: Fuzzy representation of membership function $\mu_{velocity}$

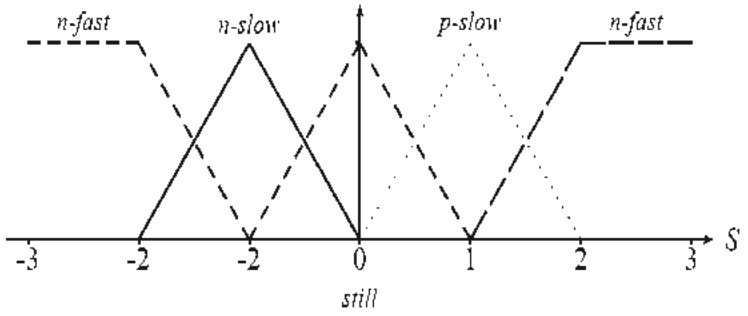


Figure 5: Fuzzy representation of membership function μ_{speed}

The collection of the fuzzy rules, which is used to control the pendulum, is given in a tabular form in Table 2. The controller rules are generally designed by an expert to achieve optimal control. For instance, the third entry in the third row reads as the rule stated in (1) while the second entry of the fourth row reads as the rule stated in (2).

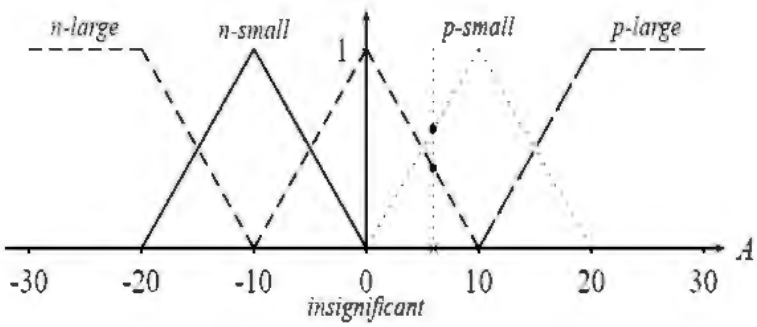
$$\text{if } angle \text{ is } insignificant \wedge velocity \text{ is } null \text{ then } speed \text{ is } still \quad (1)$$

$$\text{if } angle \text{ is } p\text{-}small \wedge velocity \text{ is } n\text{-}low \text{ then } speed \text{ is } still \quad (2)$$

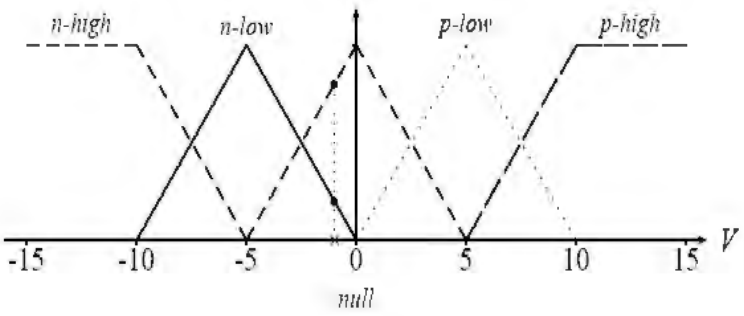
Now, we show how to apply the rules of Table 2 for specific measures of the linguistic variables *angle* and *velocity*. Assume that the value read for variable *angle* is 6° and that for variable *velocity* is -1. The values of the membership functions for variable *angle* are $\mu_{insignificant}(6) = 0.4$ and $\mu_{p\text{-}small}(6) = 0.6$ while for variable *velocity* $\mu_{null}(-1) = 0.2$ and $\mu_{n\text{-}low}(-1) = 0.8$. These points are marked on the graphs of Fig. 6.

Table 2: Fuzzy rules of the controller

rules	*n-large*	*n-small*	*insignificant*	*p-small*	*p-large*
n-high	–	–	*n-fast*	–	–
n-low	–	–	*n-slow*	*still*	–
null	*n-fast*	*n-slow*	*still*	*p-slow*	*p-fast*
p-low	–	*still*	*p-slow*	–	–
p-high	–	*p-fast*	–	–	–



(a) actual angle



(b) actual velocity

Figure 6: Membership reading for the actual values of variable *angle* and *velocity*

The rules that applies are those that have their degree of truth different from zero. So we conclude that all rules with antecedent involving the linguistic terms *insignificant* and/or *p-small* and *null* and/or *n-low* should be fired. From Table 2, we can identify that the rules that should be used are those whose consequent is framed. The four rules are also listed below in (3).

$$\begin{array}{ll}
\text{if } (angle \text{ is } insignificant) \wedge (velocity \text{ is } null) & \text{then } speed \text{ is } still \\
\text{if } (angle \text{ is } insignificant) \wedge (velocity \text{ is } n\text{-}low) & \text{then } speed \text{ is } n\text{-}slow \\
\text{if } (angle \text{ is } p\text{-}small) \wedge (velocity \text{ is } n\text{-}low) & \text{then } speed \text{ is } still \\
\text{if } (angle \text{ is } p\text{-}small) \wedge (velocity \text{ is } null) & \text{then } speed \text{ is } p\text{-}slow
\end{array} \tag{3}$$

Using Mamdani's definition of the implication operator [15], we can apply he rules in (3). The application of the selected rules yields the fuzzy sets described in Fig. 2.1, Fig. 8, Fig. 9 and Fig. 10. Always the minimum cut is used.

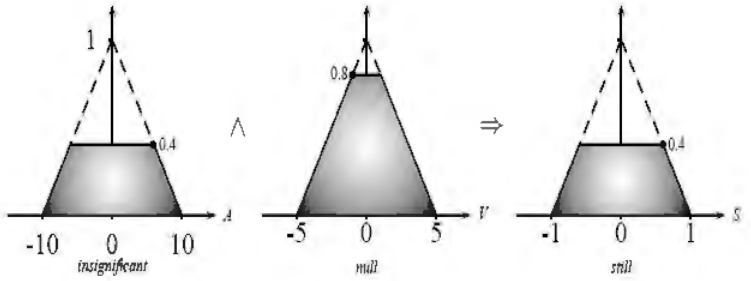


Figure 7: Representation of the result of application of the first rule in (3)

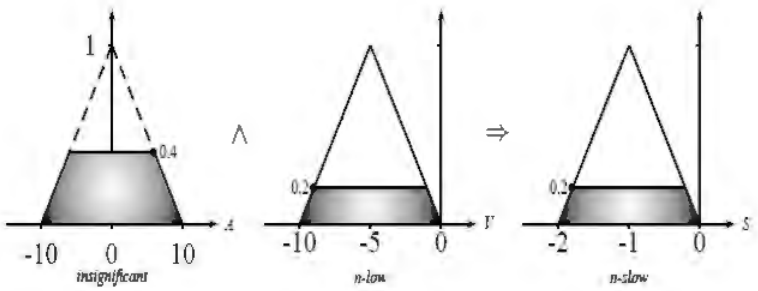


Figure 8: Representation of the result of application of the second rule in (3)

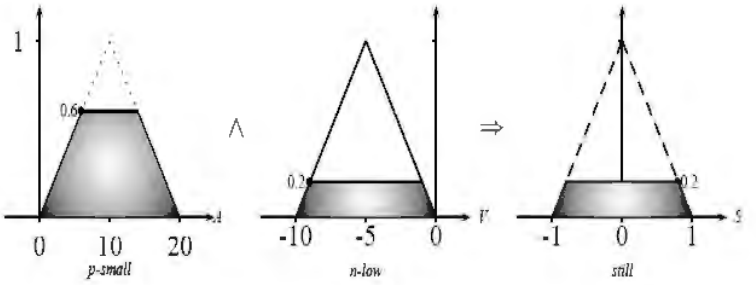


Figure 9: Representation of the result of application of the third rule in (3)

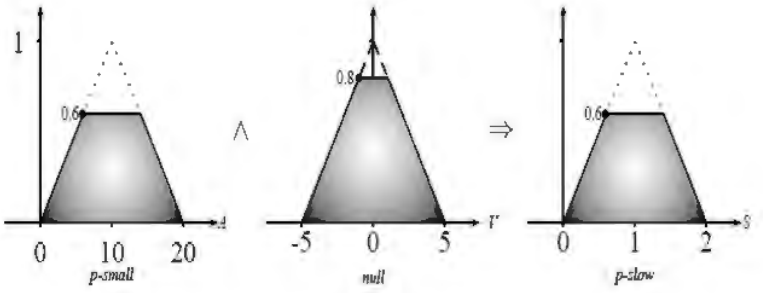


Figure 10: Representation of the result of application of the fourth rule in (3)

As the first and third rule in the selected rules yield the same consequent, i.e. *still*, we can combine them using an or-operator and therefore, the maximum cut is retained. This is depicted in Fig. 11.

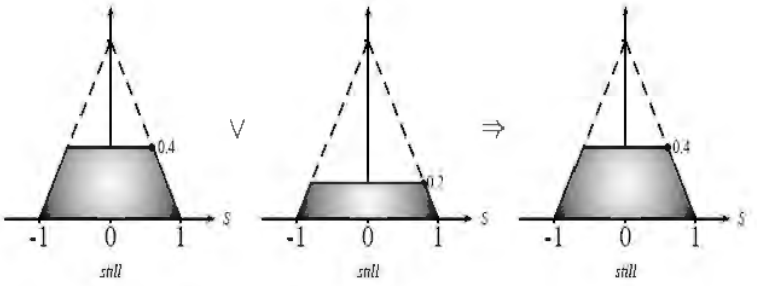


Figure 11: Combining the results of application of the first and third rule in (3)

The defuzzification of the obtained result is a very sensitive task. It depends on the nature of the process being controlled. There are several techniques for

obtaining a crisp value from a fuzzy set. However, the commonly used techniques are of two kind and are given below. Each of these techniques is illustrated in Fig. 12.

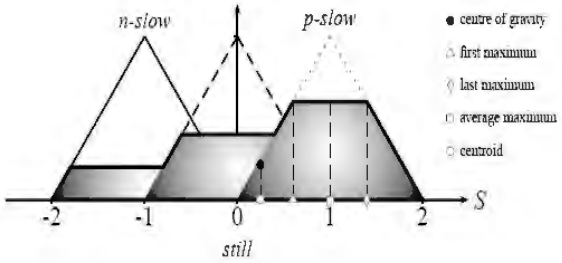


Figure 12: Composing the results of all fired rules

- the *composite moments* or *centroid*, which computes the crisp measure u as the abscissa of the center of gravity of the obtained fuzzy set. This computation for defuzzification is described in (4);

$$u = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)} \tag{4}$$

- the *composite maximum*, which is based on the values of the fuzzy set with the highest degree of truth. The composite maximum technique may use:
  - average maximum, which is the average of all the values that have the highest membership degree in the obtained fuzzy set; The computation for defuzzification is described in (5), wherein N represents the number of points with maximum membership degree in the fuzzy sets;

$$u = \frac{\sum_i \{x |\ \mu(x) = \max(x_i)\}}{N} \tag{5}$$

  - first maximum, which is the smallest value that has the highest membership degree in the obtained fuzzy set; The computation for defuzzification is described in (6);

$$u = \{x |\ \mu(x) = \max(x_i) \wedge \forall x_i, x < x_i\} \tag{6}$$

  - last maximum, which is the biggest value that has the highest degree of truth in the yield fuzzy set; The computation for defuzzification is described in (7).

$$u = \{x |\ \mu(x) = \max(x_i) \wedge \forall x_i, x > x_i\} \tag{7}$$

3 The Proposed Macro-architecture

The macro-architecture of the proposed fuzzy controller consists of three main units: *(i)* the fuzzification unit (`FU`), which is responsible for translating the input values of the system into fuzzy terms using the respective membership functions. This unit has as many `Fuzzy` blocks as required in fuzzy system model that is being implemented, i.e. one for each input variable; *(ii)* the inference unit `Inference`, which checks all the included fuzzy rules, verifying which membership function applies, and if any is so, generating its value and thus identifying the membership functions to be used in the sequel; *(iii)* the defuzzification unit (`DU`), which is responsible for translating the fuzzy terms back so as to compute the crisp value of the fuzzy controller output. The defuzzification unit includes as many `Defuzzy` blocks as required by the fuzzy system model that is being implemented, i.e. one for each output variable. The block diagram of the proposed macro-architecture is shown in the Fig. 13, wherein N and M represent the number of input and output variables, respectively.

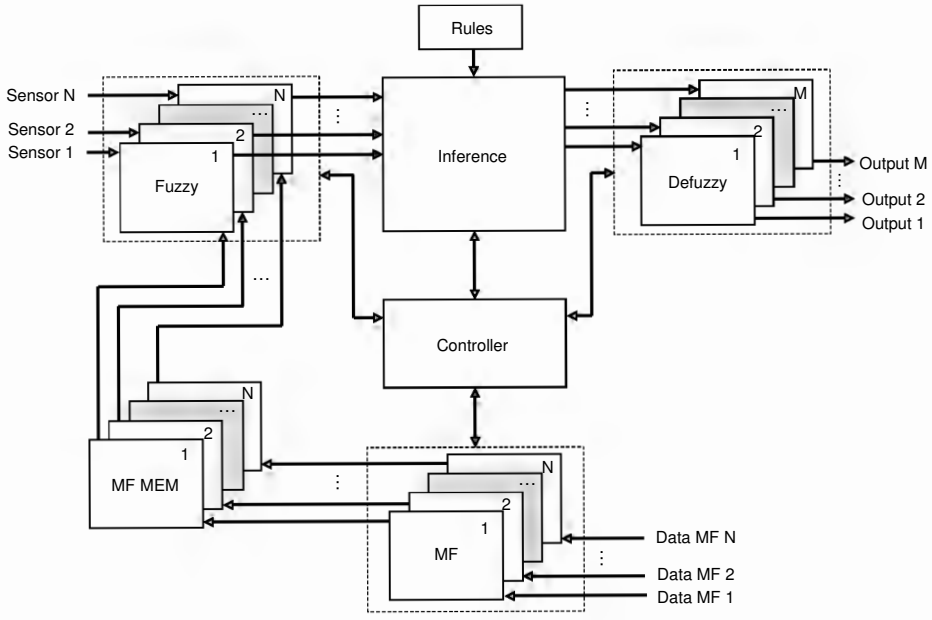


Figure 13: Macro-architecture of the designed fuzzy controller

Note that, besides the main units, the macro-architecture also includes a component that allows to compute the membership function characteristics, which are used by both the fuzzification and defuzzification units. This component will be

called membership function unit (MFU). It includes as many MF blocks as required input variable of the fuzzy model. Note that all the membership function-related data are stored in the membership function memory, called MF MEM. This memory is formed by as many memory segments as required input variables, i.e. one for each membership function used. The rules used by the inference unit are stored in a read-only memory block, called Rules. Component Controller, which in the sequel may be called *main controller*, imposes the necessary sequencing and/or the simultaneity of the required steps of the fuzzy controller via a concurrent finite state machine. More details on this are given subsequently.

The proposed fuzzy controller is designed to be generic and parametric, so it allows configuring the number of input and output variables, the number of linguistic terms used to model the membership functions and the number of inference rules, so as the fuzzy system model that is being implemented can fit in. Allowing the configuration of these parameters makes it possible, as well as easy, to adjust the controller design to any desired problem. Summing up, the main parameters of the controller are:

- N: The number of input variables and hence that of the included Fuzzy blocks;
- M: The number of output variables and hence that of Defuzzy blocks;
- P: The number of rules and thus the number of words in the rule base Rules;
- Q: The number of linguistic terms per membership functions used to model the input and output variables of the fuzzy system. Note that for the current implementation, we kept this parameter constant for all inputs and outputs. However, this can be easily altered so as to allow different models for distinct membership function.

As it can be seen in the Fig. 14, at configuration time, all the membership functions used by the controller are computed and stored in the respective MF MEM segment of the membership function memory. All the computed data will be readily available to be used by the pertinent Fuzzy and/or Defuzzy block in the fuzzification and defuzzification unit, respectively. Note that this configuration step is done only once. During the operation step, the fuzzy controller iterates the required steps, triggering the Fuzzy blocks then Inference unit then Defuzzy blocks in sequence. After that, it waits for a new set of input data to be read by the system sensors and thus arrive at the Fuzzy blocks input ports. The finite state machines that control the Fuzzy blocks all run in parallel, so do

those that control the Defuzzy blocks. Therefore, we use the Statechart description language [13] to express the hierarchical and concurrent aspects of the state machine of the overall operation of the main controller.

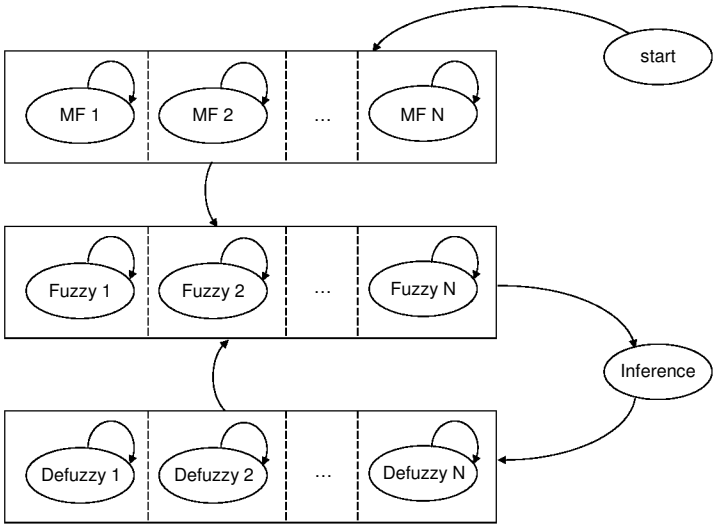


Figure 14: Hierarchy and concurrency of the main controller's state machine

In the following sections of this chapter, more light will be shed on the internal micro-architecture of the proposed design as well as the control used therein.

4 Micro-architecture of the Functional Units

In this section, we describe the micro-architecture of the main components, included in the macro-architecture of Fig. 13. These are the functional unit responsible for the computation of the member function (MF), including the memory-based component (MF MEM), the basic component responsible for the fuzzification process (Fuzzy), the component that implements the inference process (Inference) using the available rule base (Rules) and the basic component that handles the defuzzification process (Defuzzy).

In general, all blocks that perform floating-point computations include an FPU unit, which performs the main mathematical operations with simple precision (32 bits) [4]. The operations needed are addition, subtraction, multiplication and division.

4.1 Membership function unit

A membership function is viewed as a set of linguistic terms, each of which is defined by two straight lines. In the proposed design, the triangular shape is used to represent linguistic terms. Nevertheless, it is possible to adjust the design as to accept other used shapes such as trapezes and sigmoid. Fig. 15 shows a generic example of membership function with Q linguistic terms, wherein the horizontal axis x represents the controller's input, probably read from a sensor, and the vertical axis y represents the truth degree associated with the linguistic terms. This is a real value, between 0 and 1, handled as a simple precision floating-point number of 32 bits.

Linguistic terms of triangular membership function are completely defined by $MaxPoint$ and $Range$, as illustrated Fig. 15.

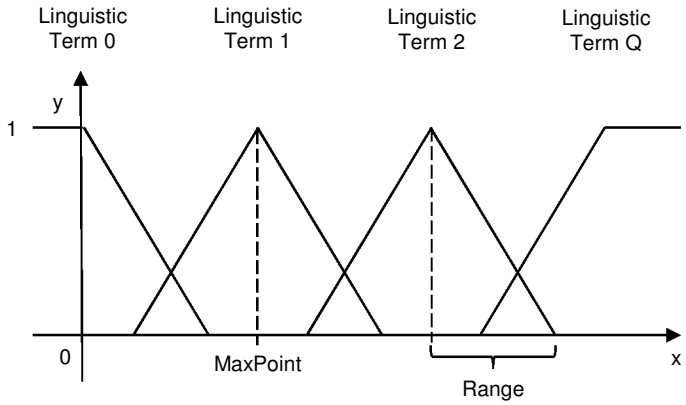


Figure 15: Membership function of Q linguistic terms

The `MF` block is designed to compute the values of any variable x, according to (8) of the two straight lines, that represent the linguistic term of the membership function. The required basic data that completely define these shapes need to be identified.

$$y = ax + b \tag{8}$$

The input data of the `MF` block are $MaxPoint$ and $Range$ for each straight line used to define the linguistic terms of the membership function. The block utilizes them and pre-compute coefficients a and b accordingly and stored them in the membership function memory segments. Three cases are possible: the leftmost linguistic term (see linguistic term 0 in Fig. 15); An in-between linguistic term (see linguistic term 1 and 2 in Fig. 15); and finally, the rightmost linguistic term

(see linguistic term Q in Fig. 15). The computation of a and b of the straight lines of the leftmost, middle and rightmost linguistic terms are defined as in (9), (10) and (11), respectively.

$$\mu_l(x) = \begin{cases} 1, & \text{if } x \leq MaxPoint \\ -\frac{1}{Range} \times x + \frac{MaxPoint}{Range} + 1, & \text{if } MaxPoint > x \\ & \geq MaxPoint + Range \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

$$\mu_m = \begin{cases} \frac{1}{Range} \times x - \frac{MaxPoint - Range}{Range}, & \text{if } MaxPoint - Range < x \\ & \leq MaxPoint \\ -\frac{1}{Range} \times x + \frac{MaxPoint}{Range} + 1, & \text{if } MaxPoint > x \\ & \geq MaxPoint + Range \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$$\mu_r(x) = \begin{cases} \frac{1}{Range} \times x - \frac{MaxPoint - Range}{Range}, & \text{if } MaxPoint - Range < x \\ & \leq MaxPoint \\ 1, & \text{if } x > MaxPoint \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

The micro-architecture of the membership function blocks `MF` is shown in Fig. 16. It uses a floating-point unit to perform the required mathematical operations. The obtained results are then stored in the `MF MEM` segments.

An `MF` block includes a controller that is implemented as a finite state machine whose state transition diagram is shown in Fig. 17. The FSM is optimized so as to include a minimum number of states. It allows to synchronize the setting up of all the linguistic terms, necessary to the complete definition of the membership function for each input variable.

In the following, we sketch how the membership function block works. When and `MF` block receives the enable command from the main controller, the state machine of Fig. 17 transits from state *start* to *test_step*, where it checks whether this stage is the first or the last straight line calculation of the membership function. If this is the case, there is no need to do anything else, because the first and last straight lines are constants, as it can be seen in Fig. 15. Therefore, the result of `FPU` block is ignored and the FSM goes to state *fpu_result*. Otherwise, i.e. if this

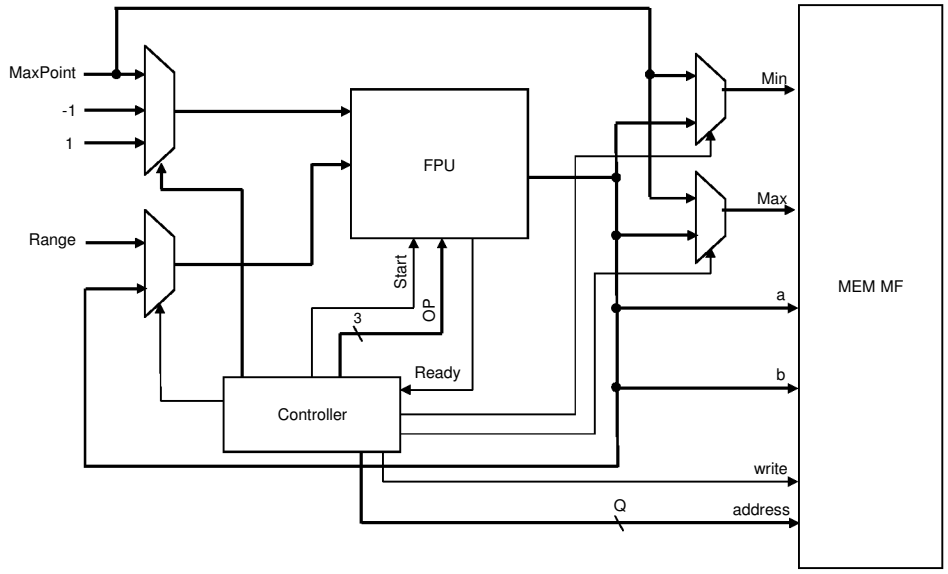


Figure 16: The micro-architecture of the membership function block

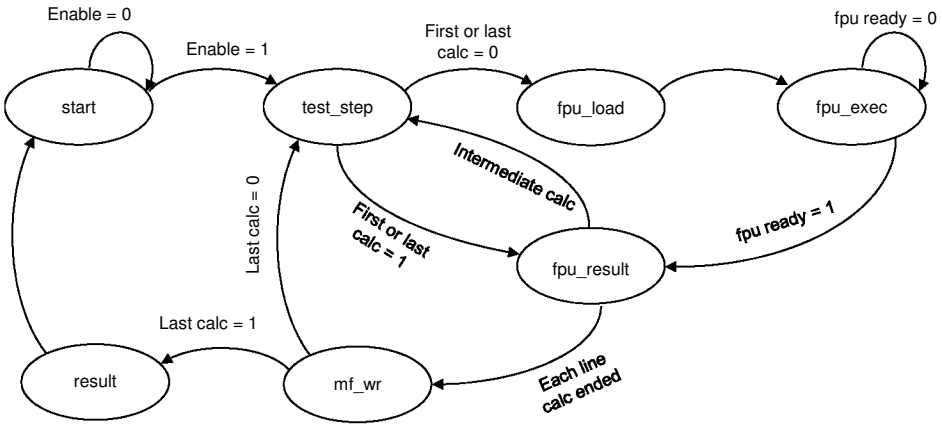


Figure 17: Membership function block state machine

is not the first or last straight line, the FSM transits to state *fpu_load*, wherein the values of $MaxPoint$ and $Range$ to be used are loaded. After that, state *fpu_exec* is entered, wherein the MF block awaits the FPU block to reach the desired result. As soon as it does, the FSM goes to state *fpu_result*, where the result is registered. Note that for each linguistic term, the MF block needs to compute four results as it will detailed later on in Section 4.2. Hence, if the computed results is not the fourth, the FSM enters state *test_step*, and iterates the process once again. Whenever all the four results are dully computed and stored, the state machine goes to state *mf_wr* where it issues the write command to MF MEM segment. The MF block iterates the same process until the last line values are written into the respective MF MEM block. Once this is done, the FSM transits to state *result* where it issues the finished signal and goes back to state *start* and waits for a new configuration stage, if any.

4.2 Membership function memory

As explained earlier, this memory block responds to write commands received from the MF block and read commands issued by the FU. Each word of this memory holds four data that allows the complete computation of the truth degree of a given linguistic term. The four-fold memory word contains:

- min: minimum limit of the straight line;
- max: maximum limit of the straight line;
- a: angular coefficient of the straight line;
- b: linear coefficient of the straight line;

So, every time that the MF block requests a memory write, this memory block register these values at an address, that represents the order number of the line within all the line that need to be processed, starting from zero. This block also allows the configuration of the number of lines that can be registered in the memory, which will depend on parameter Q, which determines the number of linguistic terms per membership function.

4.3 Fuzzification unit

The main purpose of the fuzzification unit consists of translating the input values, returned by some sensors, to linguistic terms and respective truth degrees. This is done using the data that define the membership functions, which are stored in the

MF MEM segments. Recall that for each input variable of the fuzzy system, there is a Fuzzy block associated with it.

The Fuzzy block performs the necessary computation to obtain the fuzzy version the input value. The computation consists of a comparison that may, in most cases, be followed by a multiplication then an addition, depending on the comparison result. This is repeated Q times for all the linguistic terms included in the membership function of the input variable under consideration. The Fuzzy block micro-architecture is shown in Fig. 18. It includes a Comparator that determines in which linguistic term range the input value falls, 2 sets of Q flip-flops to hold the result of the comparison. Their contents identify which linguistic terms are actually active. Note that more than one linguistic term may become active for the same given input variable. There are 2 sets because every linguistic term is represented by two straight lines. The Fuzzy block also includes an FPU block that is responsible for both the multiplication and addition. The obtained results for the 2 straight lines modeling the linguistic term are kept in two distinct 32-bit registers. These are the truth degrees once it is delivered by the FPU. The block includes 2 sets of 32-bit registers, namely TuFP 1 and TuFP 2, one for each linguistic term modeling the membership function of the input variable.

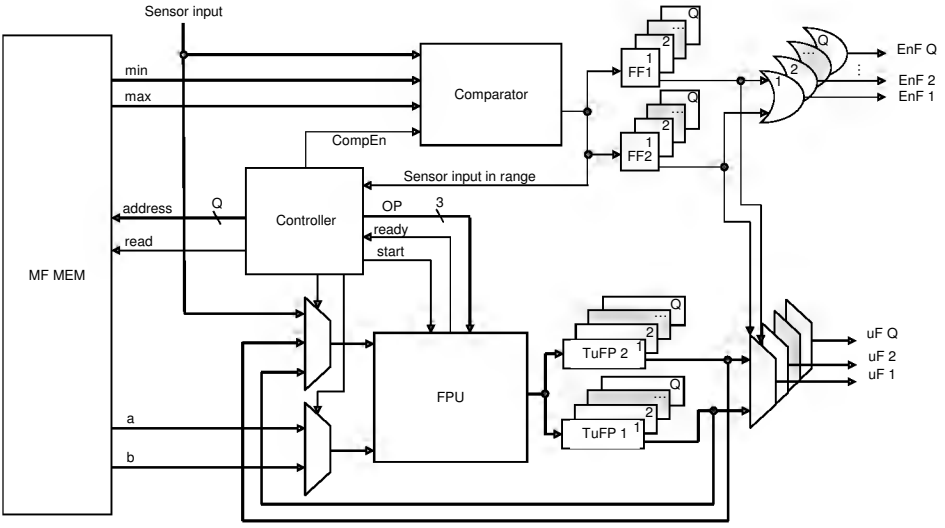


Figure 18: Fuzzy block micro-architecture

The input of a Fuzzy block are the characteristics of the linguistic terms of the membership function associated with the input variable under consideration. These characteristics are a, b, min and max stored in MF MEM segment corre-

sponding to the input variable, as explained in Section 4.2. The output of a `Fuzzy` block are: signal EnF_i, for $i = 1 \ldots Q$ bits, i.e. one for each included linguistic term and signal uF_i, for $i = 1 \ldots Q$ 32-bit floating-point values, each of which represents the truth degree of the corresponding linguistic term. Note that linguistic terms that do not apply have 0 as a truth degree. When bit EnF_i is activated, this indicates that linguistic term number i of the membership function is valid with truth degree $uF_i \neq 0$. Recall that the truth degree is the product of a and input value read by the sensor augmented by b.

Fig. 19 illustrates the dynamics imposed by the finite state machine that controls the `Fuzzy` block operation. As soon as a read request is sent to a `MF MEM` segment, the addressed memory word, containing the characteristics of the membership function, is returned to the `Fuzzy` block. The minimum and maximum values are used to check whether the input variable is within the linguistic term range, and if so, the subsequent computation of the associated truth degree is triggered. Otherwise, 0 is returned as a results.

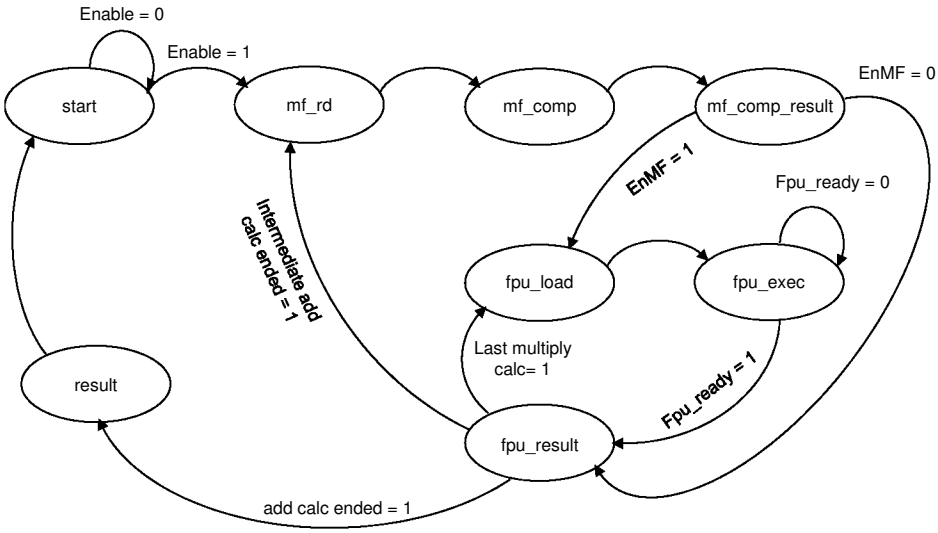


Figure 19: The transition diagram of the fuzzification controller state machine

In the following, we give an overview on how the `Fuzzy` block operates. When this block receives the enable command from the main controller (see Fig. 13), allowing it to run, the the block controlling FSM transits from state *start* to state *mf_rd*, and triggers the `MF MEM` read command. Once the memory word is received, the FSM enters state *mf_comp*, where, using the minimum and maximum values, which represent the boundaries of the straight line associated

with the linguistic term, triggers `Comparator` to perform the required comparisons to check whether the sensor input is within the boundaries of the linguistic term. Then, it shifts to state *mf_comp_result*, where it checks the result of the comparison. As every linguistic term has two lines, each result are stored in the flip-flops FF1 and FF2, as depicted in Fig. 19. When the comparison fails, i.e. input value is out if the prescribed range, the FSM goes directly to state *fpu_result*, and otherwise it shifts to state *fpu_load* to load the values to the suite of multiply-and-add, according to equation 8, shifting to *fpu_exec*. Then, the control goes to state *fpu_result* to register the obtained truth degree and after that returns to state *fpu_load*. The FSM iterates this process until there is no a calculation left to do. Whenever, the FSM enters state *fpu_result* and there still some truth degrees to be computed, it shifts to *mf_rd* to wait for a new memory word to be processed. Otherwise, i.e. the last straight line is being handled, then the FSM goes to state *result* instead, and issues the end signal to the main controller, returns to state *start* and waits for the next cycle.

Every EnF_i, for $i = 1 \ldots Q$ output signal is or-disjunction of the bits, registered by flip-flops `FF1` and `FF2`. On the other hand, every truth degree uF_i, for $i = 1 \ldots Q$ is the content of one of registers `TuFP 1` and `TuFP 2` depending on the bit value registered in flip-flops `FF1` and `FF2`. Note that truth degree uF_i will only be used in the subsequent inference stage if and only if $EnF_i = 1$.

4.4 Inference unit

The inference unit main purpose is to identify, for each one of the output variables of the fuzzy controller, the linguistic terms that are active as well as computing the associated truth degrees. It does so using the result of the fuzzification unit and the set of predefined rules that are stored in `Rules`. This is the most complicated unit in the design due to the reconfigurability characteristics of the controller, as it is explained in the remaining of this section.

Before describing the details of the inference unit, let us first introduce the structure used to format the rules of the fuzzy system. As illustrated in Section 2, a rule $\mathcal{R}$ has two defining parts: a premise $\mathcal{P}$ and a consequent $\mathcal{C}$ as described in (12), wherein $\mathcal{I}_i$, for $i = 1 \ldots N$ are the input variables and $\mathcal{T}_k^{\mathcal{I}_i}$ for $k = 1 \ldots Q$ are the linguistic terms associated to it, $\mathcal{O}_j$, for $j = 1 \ldots M$ are the output variables and $\mathcal{T}_k^{\mathcal{O}_j}$ for $\ell = 1 \ldots Q$ are the linguistic terms associated with it. Note that in general the number of linguistic terms is distinct from one variable to another. However, in this work, we assume, without loss of generality, that all the variables, both of input and output, are modeled using the same number of linguistic terms Q. A rule may check only few of of the N input variables of the

fuzzy model, and it may also, enable only few of the model output variables.

$$\begin{aligned}
\mathcal{R}: \quad & \mathcal{P} \Rightarrow \mathcal{C}, \text{ where for } j,k,\ell = 0 \ldots Q: \\
\mathcal{P} \text{ is } \quad & \mathcal{I}_0 = \mathcal{T}_j^{\mathcal{I}_0} \wedge \mathcal{I}_1 = \mathcal{T}_k^{\mathcal{I}_1} \wedge \cdots \wedge \mathcal{I}_{N-1} = \mathcal{T}_\ell^{\mathcal{I}_{N-1}} \\
\mathcal{C} \text{ is } \quad & \mathcal{O}_0 = \mathcal{T}_j^{\mathcal{O}_0} \wedge \mathcal{O}_1 = \mathcal{T}_k^{\mathcal{O}_1} \wedge \cdots \wedge \mathcal{O}_{N-1} = \mathcal{T}_\ell^{\mathcal{O}_{N-1}}
\end{aligned} \tag{12}$$

The rule base memory `Rules` has a word size that allows to store one rule. All the rules of the model have the same structure. They include all the input output variables. When a variable is not checked or inferred, the all the linguistic terms are checked off. The binary format of a rule as it is handled in the design is depicted in Fig. 4.4. It includes Q bits for each input and output variables, one bit for every allowed linguistic term. Hence, a rule occupies a total of $(N + M) \times Q$ bits.

The rule base memory `Rules` has P rules and thus will have $P \times (N+M) \times Q$ bits. A request to read `Rules` at some address will deliver the whole rule. The bits of the premise of are used, first of all, to check whether the rule under consideration can be fired, and if so, to trigger the computation of the truth degrees of the associated the consequent linguistic terms of the checked output variables.

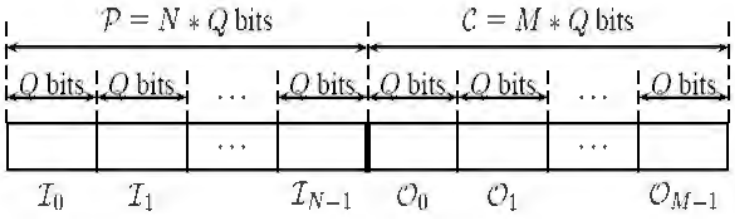


Figure 20: Inference rule format

A given rule fires when signal EnF_i, as delivered by the `FU`, for every checked of linguistic term of every input variable of the premise part of the rule under consideration is set. Furthermore, every linguistic term of any output variable that is checked in the consequent part of a fired rule need to be reported to the defuzzification unit `FU`. Notes that there are at most M, one for each output variable. Besides this, `FU` needs also to receive the truth degree for each of these checked terms.

The truth degree of an output variable linguistic term is the smallest truth degree, considering all those associated with the input variable linguistic terms in the premise part of the fired rule. When the same output variable linguistic term appears on two or more fired rules, the highest truth degree is used. Thus this done

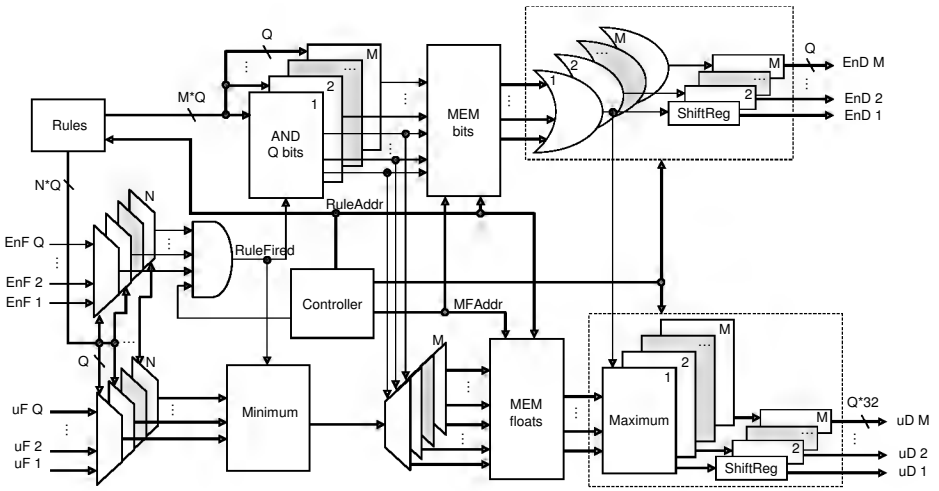
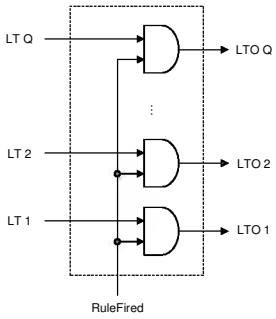


Figure 21: Inference block micro-architecture

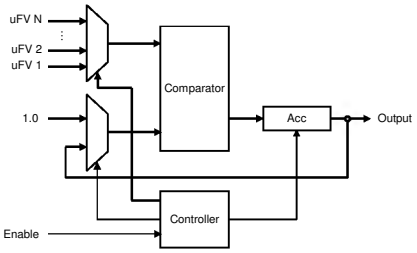
considering all the rules that fires.Recall that the truth degree of the input variable linguistic terms are provided by the `FU`.

Fig. 21 shows the micro-architecture of the `Inference` block. Its inputs consist of the Q flags EnF_i, for $i = 1 \ldots Q$ and the corresponding Q truth degrees uF_i, for $i = 1 \ldots Q$, which are the resulting output of `FU`, as described in Section 4.3. Its outputs are a set of M Q-bit signals EnD_i, for $i = 1 \ldots M$, that identify the linguistic terms that were inferred and their respective truth degrees uD_i, for $i = 1 \ldots M$, which are signals of $Q \times 32$ bits. The AND gate determines wither the current rule can be fired. The M `ANDQbits` components is a simply AND-arrays, is described in Fig. 22(a). In this design, the process of min-max inference is used. So, components `Minimum` and `Maximum` return the smallest of N floats and the highest of M floats, respectively. Their internal structure is given in Fig. 22(b) and Fig. 22(c), respectively. Note that initially the input is compared with 1.0 and after that with the minimum selected so far. Similarly, constant 0.0 is instead.

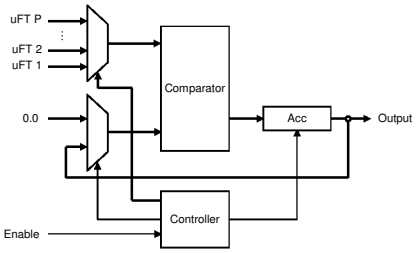
The `Inference` includes three memory blocks: the rule base `Rules`, a truth degree memory `MEM floats` and a bit memory `MEM bits`. Their respective structure are shown in Fig. 23(a) and Fig. 23(b). A write request of `MEMbits` or `MEMfloats` stores a row, i.e. $M \times Q$ or $M \times Q \times 32$ bits, respectively, while a read request releases $P \times M$ and $P \times M \times 32$ bits. Assume that the ith rule of `Rules` has fired. So, the data stored in ith row of `MEMbits` consists of the consequent part of that rule, and otherwise, all the bits are reset. Similarly, the data stored in ith row of `MEMfloats` consists of the obtained minimum truth



(a) ANDQbits

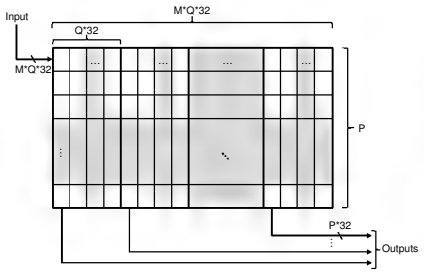


(b) Minimum

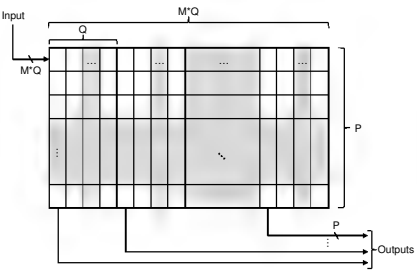


(c) Maximum

Figure 22: Internal structure of the auxiliary components: selection of the consequent part of the fired rule, the minimum and maximum truth degrees



(a) MEMfloats



(b) MEMbits

Figure 23: Internal structure of the auxiliary memories for inferred linguistic terms and their respective truth degrees

degree for that rule duplicated in all columns where a linguistic term is checked in the rule consequent.

As we can see in the Fig. 24, the state machine was also optimized to have the minimum number of states possible. This was done bearing in mind as to allow the reconfiguration of the number of the linguistic terms of the membership functions, rules, input output variables. Basically, there are two loops in the control imposed by the FSM. The first loop allows reading the rules one after the other and identifying the minimum of the associated truth degrees. The results of each iteration are stored in a given row of `MEMfloats`. The second loop uses the content of `MEMfloats` column after column to identify the maximize truth degree for all linguistic terms that are associated with more than one.

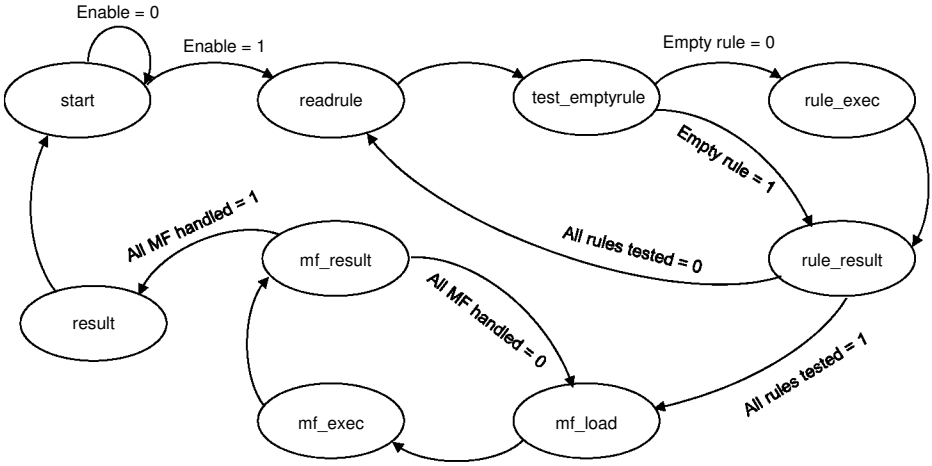


Figure 24: The transition diagram of the inference controller state machine

In the following we sketch how the operation of the inference block is controlled. When the `Inference` block receives the enable command from the main controller allowing it to run, the state machine transits from state *start* to state *readrule*, where a specified rule to be executed is selected and read from memory `Rules`. Then, the control shifts to state *test_emptyrule*, where the rule loaded is checked whether it is empty. If so, the FSM goes to state *rule_result*. Otherwise, it enters state *rule_exec*. As shown in Fig. 21, the information of the rule premise is dispatched so as to control the operation of the two set of multiplexers. Note that there is two multiplexer for each input variable: one the flags and the other for the truth degrees. In state *rule_exec*, for each fired rule, the `ANDQbits` and `Minimum` operates simultaneously and the obtained results are stored in `MEMbits` and `MEMfloats` respectively. In the latter, before writing

occurs, the results go through the set of M demultiplexers in order to associate the selected smallest truth degree to each and every one of the linguistic terms of the consequent part of the rule under consideration. The so far described process is iterated for all existing rule in the rule base. So, if the handled rule is not the last, state *readrule* is entered again and the precess is repeated. Otherwise, i.e., the last rule was processed, the FSM goes to state *mf_load* where the second loop initiates. In this state, the truth degrees of the same linguistic term of all rules are read from `MEMfloats` so as to provide the input to the `Maximum` component, which operates as soon as the FSM enters state *mf_exec*. This process is iterated Q times, which allows for the processing of the content of `MEMfloats`. After that, state *mf_result* is entered to shift the result in the shift register at the end of the chain in Fig. 24. IF there are still M columns to handle, the FSM shifts back to state *mf_load*. Otherwise, it enters to state *result*, generating the end signal to the main controller and going back to state *start* to wait for the next cycle.

4.5 Defuzzification unit

The defuzzification unit main purpose is to compute the crisp value of the output variables, given the fuzzy linguistic terms and their corresponding truth values, as identified and computed by the inference unit. The centroid, as defined in (13), is used to perform the defuzzification process. Recall that uD_i for $i = 1 \ldots Q$ are the truth degrees of the linguistic terms associated with the output variable $\mathcal{O}_i$. This method computes the geometric center of the output membership function, considering the activated linguistic terms received from the inference block together with their respective truth degrees. The computation is done according to the steps of Algorithm 1.

$$\mathcal{O}_i = \frac{\sum_{j=0}^{Q} uD_j^i \times MaxPoint_j^i}{\sum_{j=0}^{Q} uD_j^i} \tag{13}$$

Fig. 26 shows the state transition diagram of the FSM that controls the `Defuzzy` block operation. Hereafter we sketch the main steps of this control. When this block receives the enable command from the main controller allowing it to run, the FSM goes from state *start* to state *test_empty*, where the possibility of all possible linguistic terms are not enabled. If so, the FSM enters state *result*, which allows the `Defuzzy` to return 0 as output result. In this case, it goes immediately to state *fpu_result*, because there is nothing to compute. Otherwise, i.e if at least one linguistic term for the output variable that is being processed is set, then the

Algorithm 1 Steps for the computation of the centroid of output $\mathcal{O}_i$

Require: bits $EnD_i^j,\ j = 1 \ldots Q$ and and floats $uD_i^j,\ j = 1 \ldots Q$ for output variable $\mathcal{O}_i$;
$R_1 \Leftarrow 0$;
if $EnD = 00 \ldots 0$ **then**
for $j := 1\ to\ Q$ **do**
if $EnD_j^i = 1$ **then**
$R_2 \Leftarrow uD_j^i \times MaxPoint_j^i$;
$R_1 \Leftarrow R_1 + R_2$;
end if
end for
$R_2 \Leftarrow 0$;
for $j := 1\ to\ Q$ **do**
if $EnD_j^i = 1$ **then**
$R_2 \Leftarrow R_2 + uD_j^i$;
end if
end for
$R_1 \Leftarrow R_1/R_2$;
end ifreturn R_1;

FSM goes to *fpu_load*, where the control enables that the values of the specific computation to be loaded and thereafter goes to state *fpu_exec*, where the computation described in Algorithm 1 is executed. (For the sake of clarity, the details of the necessary three operations are omitted in this description.) Once the computation performed one iteration is completed, the FSM shifts to state *fpu_result*, where it checks whether all EnD_i and respective uD_i for $i = 1 \ldots Q$ have been considered. If not, the FSM goes back to state *test_empty* and iterate repeats the same process. Otherwise, the result is readily registered and available in register R_1 and so, the FSM enters state *result*, generating the main controller's output value and the issuing a $done$ signal and the block becomes ready again to operate from the start.

5 Simulation Results

The proposed design for the fuzzy controller was modeled using VHDL [19]. The functional simulation of the obtained model was done using ModelSim from Mentor Graphics [28]. Needless to point out that many controller configurations

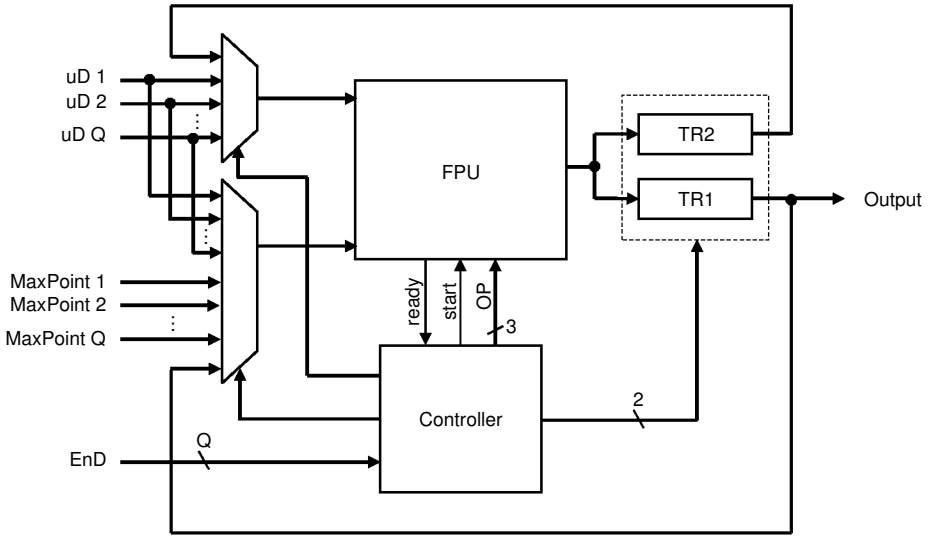


Figure 25: Defuzzification block micro-architecture

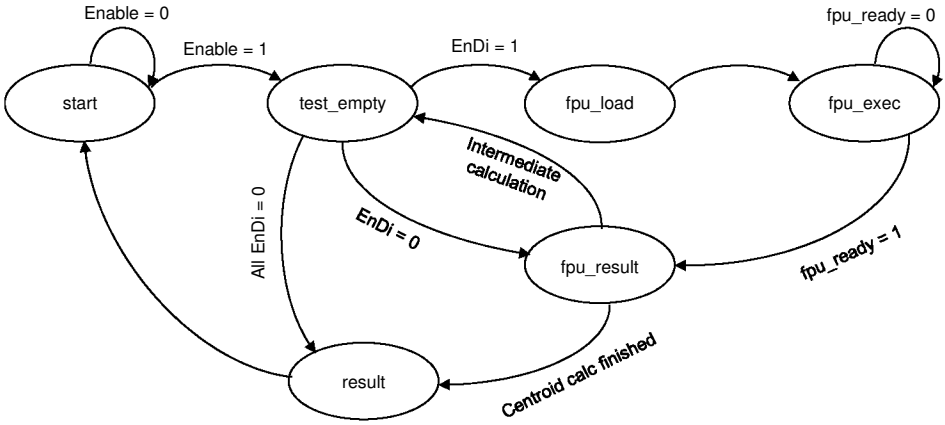


Figure 26: Transition diagram for state machine that controls the defuzzification process

were simulated and all performed correctly, delivering the right results according to the fuzzy model used.

For validation and completeness sake, we show, in the remaining of this section, some snapshots of the simulation of the main stages of the fuzzy controller: fuzzification, inference and defuzzification. This is done for the same example of the pendulum, used in Section 2. Recall that that the fuzzy controller requires 2 input variables, which are A (angle) and V (velocity), and a single output variable S (speed). All variables are modeled using 5 linguistic terms. (Refer to Fig. 3, Fig. 4 and Fig. 5 for details about the membership functions of these variables.) The rule base includes 11 among 25 possible rules, as shown in Table 2.

The control situation, we show in the simulation snapshots is related to the case wherein $A = 6$ and $V = -1$. For the sake of clarity, this case is the same reported in Section 2. The controller took 430 clock cycles to execute all the computation to the membership function while the fuzzification, inference followed by the defuzzification process lasted 651 clock cycles. Of course, if the number of rules or linguistic terms of the membership functions increase, this duty cycle will also increase accordingly. The clock cycle is defined by the floating-point unit used IP [4], which can run at a frequency of 100 MHz.

Fig. 27 shows the first snapshot of the simulation process of the fuzzification process of the input values while Fig. 28 shows the second and last snapshots of the simulation. This shows a simulated test of the fuzzification block configured with 5 linguistic terms, so after the membership function block has completed, all data are stored in the `MF MEM` segments. Once the input value is ready in signal $SensorInput$, signal $EnFuzzy$ enables the fuzzification process. Signal $MFAddr$ indicates in the address of the straight line characteristics (a, b, min and max) of the linguistic term in the membership function memory `MF MEM`. Observe that signal $inRange$ goes high when $MFAddr$ is 5 and 6, which correspond to right-hand side straight line of linguistic term $insignificant$ of A and the left-hand side straight line of V (see Fig. 3). Signal EnF for variable A is 00100 (see Fig. 27) when straight line number 5 is checked and 01100 (see Fig. 28) when number 6 is checked. All calculations are done using the `FPU` block, which receives its inputs via signals $fpuInput1$, $fpuInput2$ and the code of the operation that must be performed via signal fpu_op at that moment. The actual operation execution starts when signal fpu_start is triggered and ends when flag fpu_ready comes. Observe that the the `FPU` performed two operations (1 multiplication and 1 addition/subtraction) for each selected straight line. The output of `FPU` is shown in signal $fpuOutput$. Using Hexadecimal, the first truth degree computed is `3ECCCCCC` and the second is `3F19999A`, which correspond exactly to $0.39999998 \approx 0.4$ and 0.6. The current state of the fuzzification FSM

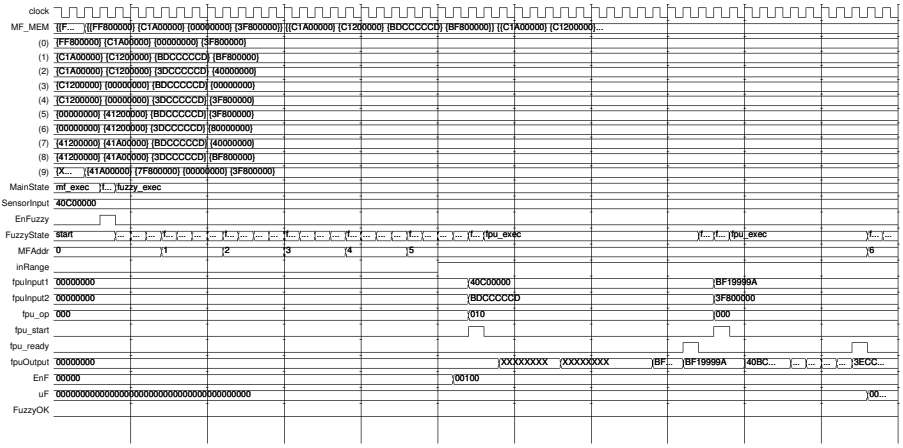


Figure 27: Simulated operation of the fuzzification block – Snapshot 1

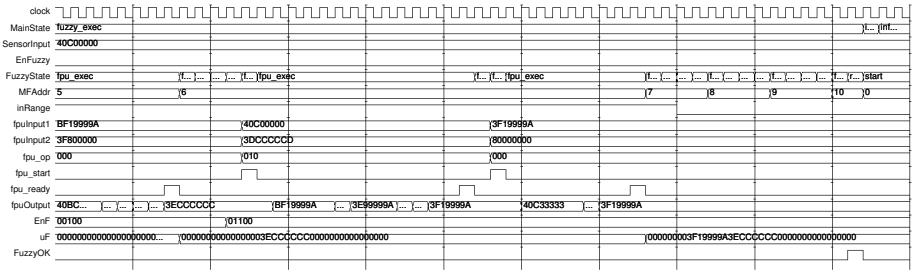


Figure 28: Simulated operation of the fuzzification block – Snapshot 2

is represented by the signal $FuzzyState$ while that of the overall fuzzy controller is represented by $MainState$.

The snapshots of the fuzzification process regarding the input variable V has been omitted. Recall that this process runs simultaneously with the one regarding variable A. For variable V straight lines 3 and 4 would be used and the corresponding truth degrees are `3F4CCCCD` and `3E4CCCCD`, which correspond exactly to 0.8 and 0.2, respectively. Note that the signal EnF for variable V would be 01100. The reader can verify this in Fig. 29, which shows the beginning of the simulation snapshot of the inference process. It utilizes 11 from the 25 possible rules. It draws its input from 2 fuzzification blocks and deliver its results to 2 defuzzification blocs. Fig. 30 and Fig. 31 show the second and third (and last) snapshots. Observe that signal `Rules` shows the configuration of the premise and consequent parts of the rules that will be used in the inference process. Signal $EnInf$ enables the inference process. Signals EnF and uF are the result of the

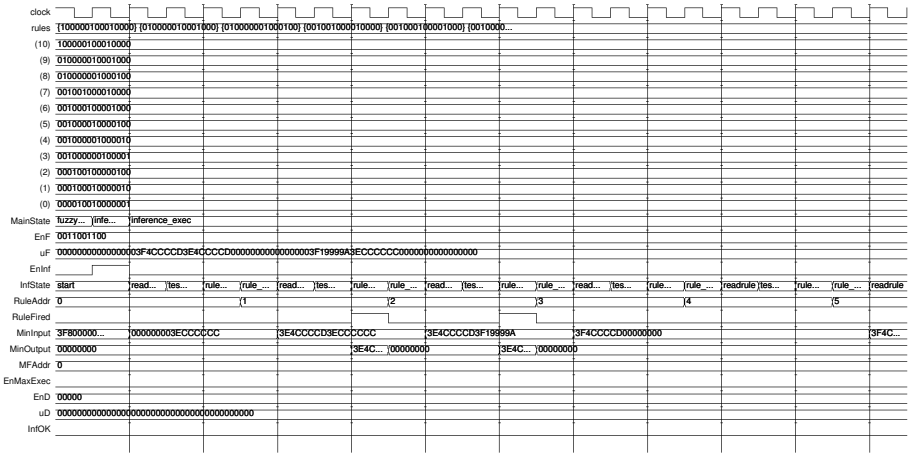


Figure 29: Simulated operation of the inference block – Snapshot 1

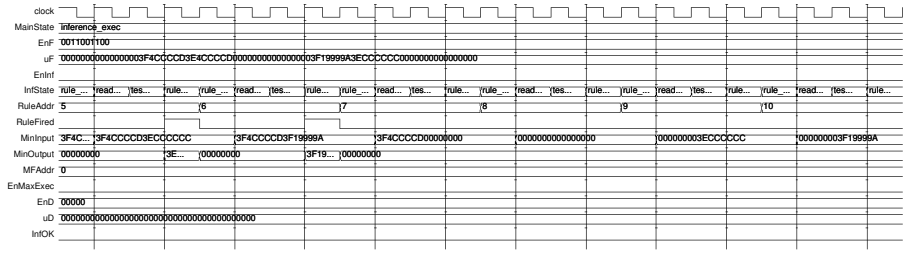


Figure 30: Simulated operation of the inference block – Snapshot 2

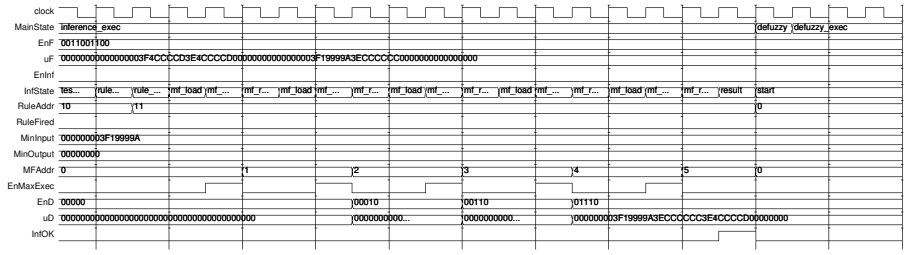


Figure 31: Simulated operation of the inference block – Snapshot 3

2 fuzzification processes. The current state of the inference FSM is represented by the signal $InfState$ while that of the overall fuzzy controller is, as before, represented by $MainState$. Signal $RuleAddr$ samples the rule number that is being checked. Observing signal $RuleFired$, note that rules 1, 2, 5 and 6 fire. Value `3F800000` is constant 1 used as one of the input of the `Minimum` block in the first iteration. The truth degrees obtained by this block when are `3F4CCCCD`, `3F4CCCCD`, `3ECCCCCC` then `3F19999A` (i.e. 0.2, 0.2, 0.4 then 0.6) as shown in signal $MinOutput$. These truth degrees are obtained for rule 1, 2, 5 and 6 (see signal $RuleFired$) respectively. After getting the minimum truth degree for each fired rule, it is necessary to compute the maximum truth degree associated with each linguistic term, as enabled by signal $EnMaxExec$ (see Fig. 31). When `3ECCCCCC` and `3F4CCCCD` are compared, the output result is `3ECCCCCC`. For the other two truth degrees, the linguistic term was inferred by only one rule. The final result of the inference process is given in signal EnD wherein 3 linguistic terms were inferred with truth degrees as shown in signal uD of Fig. 31.

Fig. 32 shows the first snapshot of the defuzzification process, configured to work with 5 linguistic terms while Fig. 33 shows the second and last snapshot of the simulation. Signal $EnDefuzzy$ enables the process. So, it starts the computation of the crisp value of the output variable, based on the identified fuzzy linguistic terms checked and their respective truth degrees. it does so using the `FPU` block, which, as in the fuzzification process, receives its input values in signals $fpuInput1$ and $fpuInput2$ and the operation that should execute in signal fpu_op. The operation execution start is can be identified by a pulse in signal fpu_start and the end occurs when a pulse appears on signal fpu_ready. For instance, in Fig. 32, the first operation is a multiplication of truth degree `3E4CCCCD` = 0.2 by $MaxPoint$ `BF800000` = -1 as shown in signals $fpuInput1$ and $fpuInput2$ respectively. After the operation is completed the result is `BE4CCCCD` = -0.2, as shown in signal $fpuOutput$. The current state of the FSM is represented by signal $DefuzzyState$ while that of the overall fuzzy controller is, as before, represented by $MainState$. For this example, the $MaxPoints$ are `40000000` = 2.0, `3F800000` = 1.0, `00000000` =0.0, `BF800000` = -1.0 and `C0000000` = -2.0, as it can be seen in signal $MaxPOints$ of Fig. 32. The weighted sum would then be $1.0 \times 0.6 + (-1.0) \times 0.2 = 0.4$ and the sum of all truth degrees are $0.2 + 0.4 + 0.6 = 1.2$. In Fig. 33, observe that the controller final output is the result of the division of `3ECCCCCE` = $0,40000004$ by `3F99999A` = 1.2, i.e. `3EAAAAAB` = 0.33333334, which is the crisp value of output variable S given by the fuzzy controller in response to input values $A = 6$ and $V = -1$. The infinitesimal and thus harmless difference in the final digit is due to the simple floating-point precision representation.

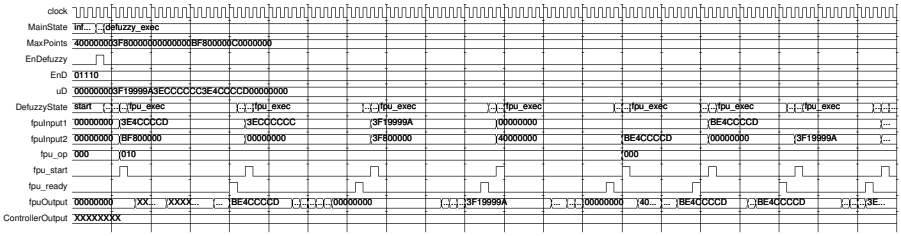


Figure 32: Simulated operation of the defuzzification block – Snapshot 1

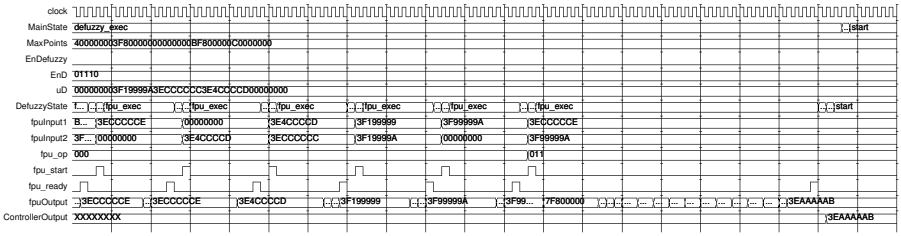


Figure 33: Simulated operation of the defuzzification block – Snapshot 2

The proposed design has some constraints to guarantee the delivery of correct results: The minimal number of linguistic terms in the membership function must be at least 2, which is the case in any fuzzy model. Each linguistic term will always be represented by an isosceles triangle, as the range is set to be the same to both sides at the membership function, as seen in Fig. 15. Nevertheless, the maximum amount of rules is not pre-determined, so the expert can use any number of rules that can be accommodated within the reconfigurable device.

6 Conclusions

The proposed design for fuzzy controllers is a applicable to almost any applications in the industry that do not have a prescribed solution. The proposed architecture is parametric so that any number of inputs, outputs and rules can be accommodate with no extra effort.

The next steps in the design of this controller are the design synthesis into a reconfigurable device, such as FPGAs and the evaluation of the required area as well as the operation frequency. This will allow us to assess the performance of any given application, wherein the fuzzy controller, proposed in this chapter, is used. Also, we intend to investigate the generalization of the design so that to allow the use of trapezoidal and sigmoid the membership functions.

Acknowledgments

We are grateful to FAPERJ (Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro, http://www.faperj.br) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico, http://www.cnpq.br) for their continuous financial support.

References

[1] Baldwin, J. F., Fuzzy logic and fuzzy reasoning, In Fuzzy Reasoning and Its Applications, E.H. Mamdani and B.R. Gaines (Eds.), London Academic Press, 1981.

[2] Bandler, W. and Kohout, L. J., Semantics of implication operators and fuzzy relational products, In Fuzzy Reasoning and Its Applications, E.H. Mamdani and B.R. Gaines (Eds.), London Academic Press, 1981.

[3] Bernstein, D., Control tutorial for MatLab: inverted pendulum, University of Michigan, USA, www.engin.umich.edu/group/ctm/examples/pend/invpen.html, December 2004.

[4] Jidan Al-E., Float Point Unit Manual, 2006.

[5] Daijin K., An Implementation of Fuzzy Logic Controller on the Reconfigurable FPGA System, IEEE Trans. on Industrial Electronics, vol. 47, no. 3, 2000.

[6] Diao, Y., Hellerstein, J. and Parekh, S., Using fuzzy control to maximize profits in service level management, IBM Systems Journal, vol. 41, no. 3, pp. 403–420, 2002.

[7] Eschbach, M. and Cunnyngham, J., The logic of fuzzy Bayesian influence, International Fuzzy Systems Association Symposium of Fuzzy information Processing in Artificial Intelligence and Operational Research, Cambridge, England, 1984.

[8] Esragh, F. and Mamdani, E.H., A general approach to linguistic approximation, in Fuzzy Reasoning and Its Applications, E.H. Mamdani and B.R. Gaines (Eds.), London Academic Press, 1981.

[9] Franke, K., Köppen, M. and Nickolay, B., Fuzzy image processing by using Dubois and Prade fuzzy norm, In Proceedings of 15th International Conference on Pattern Recognition, Barcelona, Spain, pp. 518–521, 2000.

[10] Fox, J., Towards a reconciliation of fuzzy logic and standard logic, International Journal of Man-Machine Studies, vol. 15, pp. 213–220, 1981.

[11] Ghidary, S., Hattori, M., Tadokoro, S. and Takamori, T., Multi-modal human robot interaction for map generation. Proceedings of IEEE International Conference on Intelligent Robots and Systems, pp. 2246–2251, 2001.

[12] Haack, S., Do we need fuzzy logic?, International Journal of Man-Machine Studies, vol. 11, pp. 437–445, 1979.

[13] Harel, D., A Visual Formalism for Complex Systems, Science of Computer Programming , pp. 231-274, 1987.

[14] Magdalena, L. and Velasco, J.R., Fuzzy Rule-Based Controllers that Learn by Evolving their Knowledge Base, In Herrera F. and Verdegay J.L. (Eds.), Genetic Algorithms and Soft Computing, Physica-Verlag, pp. 172–201, 1996.

[15] Mamdani, E. H. and Assilian, S., An experiment in linguistic synthesis of fuzzy controllers, International Journal of Man-Machine Studies, no. 7, pp. 1–13, 1975.

[16] Mamdani, E. e Pappis, C., A fuzzy logic controller for a traffic intersection, IEEE Trans. on Systems, Man and Cybernetics 22 (6):1414-24, 1977

[17] Masmoudi N., Hachicha M., Kamoun L., Hardware Design of Programmable Fuzzy Controller on FPGA, Laboratory of Computer Science and Industrial Electronic of Sfax.

[18] McKenna, M. and Wilamowski, B., Implementing a Fuzzy System on a Field Programmable Gate Array Fuzzy Sets and Systems, University of Wyoming and University of Idaho.

[19] Navabi, Z., VHDL: Analysis and Modeling of Digital Systems, McGraw Hill, Second Edition, 1998.

[20] Nedjah, N., Mourelle, L.M., Fuzzy Systems Engineering, Advanced Studies in Fuzziness and Soft Computing, Springer, vol. 181, 2005.

[21] Poorani, S., Urmila Priya, T.V.S., Udaya K. and Renganarayanan, S., FPGA based Fuzzy Logic Controllers for Electric Vehicle, Journal of the Institution of Engineers, Singapore, vol. 45, no. 5 2005.

[22] Rachel, F. M. (2006), Proposta de um controlador automático de trens utilizando lógica nebulosa preditiva, M.Sc. Dissertation, University of São Paulo, Brazil.

[23] Radecki, T., An evaluation of the fuzzy set theory approach to information retrieval, in R. Trappl, N.V. Findler, and W. Horn (Eds.), Progress in Cybernetics and System Research, vol. 11, Proceedings of a Symposium Organized by the Austrian Society for Cybernetic Studies, Hemisphere Publishing Company, NY. 1982.

[24] Shim, D.H., Koo, T.J., Hoffmann, F. and Sastry, S.S., A comprehensive study of control design for an autonomous helicopter, Proceedings of 37th. Conference on Decision and Control, Tampa, FL, pp. 3653–3658, 1998.

[25] Sulaiman, N., Obaid, Z. A., Marhaban, M. H. and Hamidon, M. N., FPGA-Based Fuzzy Logic: Design and Applications: a Review, IACSIT International Journal of Engineering and Technology Vol.1, No.5, December, 2009.

[26] Takemura, R. Y., Lógica Difusa, url: http://www.din.uem.br/ia/controle/fuz_prin.htm.

[27] Umbers, I.G. and King, P.J., An analysis of human decision-making in cement kiln control and the implications for automation, International Journal of Man-Machine Studies, vol. 12, pp. 11–23, 1980.

[28] Xilinx, Inc., Foundation Series Software, `http://www.xilinx.com`, 2011.

[29] Zadeh, L.A., Fuzzy Sets , Journal of Information and Control, Vol. 8, pp. 338-353, 1965.

[30] Zadeh, L.A., Fuzzy algorithms, Information and Control, vol. 12, pp. 94–102, 1968.

[31] Zadeh, L. A., Making computers think like people, IEEE Spectrum, no. 8, pp. 26–32, 1984.

[32] Zadeh, L.A., Fuzzy Logic, IEEE Computer Journal, vol. 1, no. 83, p. 18, 1988.

[33] Zimmermann, H.J. and Zysno, P., Latent connectives in human decision making, Fuzzy Sets and Systems, vol. 4, pp. 37–51, 1980.

[34] Zhang, J. and Knoll, A., Designing Fuzzy Controllers by Rapid Learning, Fuzzy Sets and Systems, 101, pp. 287–301, 1999.

[35] Wachs, J., Stern, H. and Edan, Y., Parameter Search For An Image Processing Fuzzy C-means, Proceedings of IEEE International Conference on Image Processing, Barcelona, Spain, vol. 3, pp. 341–344, 2003.

The papers presented in this Volume 2 constitute a collection of contributions, both of a foundational and applied type, by both well-known experts and young researchers in various fields of broadly perceived intelligent systems.
It may be viewed as a result of fruitful discussions held during the Tenth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets (IWIFSGN-2011) organized in Warsaw on September 30, 2011 by the Systems Research Institute, Polish Academy of Sciences, in Warsaw, Poland, Institute of Biophysics and Biomedical Engineering, Bulgarian Academy of Sciences in Sofia, Bulgaria, and WIT - Warsaw School of Information Technology in Warsaw, Poland, and co-organized by: the Matej Bel University, Banska Bystrica, Slovakia, Universidad Publica de Navarra, Pamplona, Spain, Universidade de Tras-Os-Montes e Alto Douro, Vila Real, Portugal, and the University of Westminster, Harrow, UK:

Http://www.ibspan.waw.pl/ifs2011

The consecutive International Workshops on Intuitionistic Fuzzy Sets and Generalized Nets (IWIFSGNs) have been meant to provide a forum for the presentation of new results and for scientific discussion on new developments in foundations and applications of intuitionistic fuzzy sets and generalized nets pioneered by Professor Krassimir T. Atanassov. Other topics related to broadly perceived representation and processing of uncertain and imprecise information and intelligent systems have also been included. The Tenth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets (IWIFSGN-2011) is a continuation of this undertaking, and provides many new ideas and results in the areas concerned.

We hope that a collection of main contributions presented at the Workshop, completed with many papers by leading experts who have not been able to participate, will provide a source of much needed information on recent trends in the topics considered.

ISBN-13 9788389475411