# Combinatorial Fuzzy Logic Controller Design

Arturo Tellez, Luis Villa, Heron Molina,
Oscar Camacho and Romeo Urbieta

Centro de Investigación en Computación, Instituto Politécnico Nacional, Juan de Dios Batiz Ave. s/n, Nueva Industrial Vallejo, Mexico City, Mexico
http://www.microse.cic.ipn.mx/

**Abstract.** This paper presents the architecture development of a Fuzzy Logic Controller (FLC), using combinatorial design implemented on a Field Programmable Gate Array (FPGA). This architecture is based on combinatorial basic modules that enable to increase and improve the entire system performance, by means of replication technique, which is widely used in computer architecture, and help to fit the particular application needs. There have been so many FLC implementations since the first hardware one appeared [4] which used complex designs with sequential circuits because of the high hardware resource and delay time costs about combinatorial design. But recent FPGA technology let us use fast combinatorial circuits for complex designs with parallelism for increasing the FLC performance and it is possible to take it up again as a practical way to build FLC for any process, approaching the fast prototyping advantages and easing the scaling to increase the control accuracy.

## 1 Introduction

There has been a large increment in the use of FLC in many science and industry fields early, so the implementations in several technologies, from the use of fuzzy microcontrollers until reprogrammable ways, as FPGA means.

An FLC can be implemented in software easily and executed in a microprocessor, a microcontroller, or a general purpose computer. Though software- based FLC are cheaper and flexible, there are some difficulties when control systems require high data processing.

The use of FPGA has been profitable when we talk about versatility to make any digital design by means of costs and design time. With fuzzy logic application in high velocity systems of industry, especially in real time systems, as image processing, robotics, automotive, and other industries, it is necessary to develop new faster implementations. FPGA came from the Applied Specific Integrated Circuit (ASIC) industry as an economic and flexible option, for processing data at competitive velocity and opening a wide door for researchers to make their own designs in short time.

In principle, the implementation of FLC is not based on the mathematic model of the plant, but this kind of system is very effective to control a process where the transfer function is not known, instead the control action is based on the extern influence and simple decisions based on a knowledge base acquired with experience, the same


Advances in Computer Science and Engineering.
Research in Computing Science 42, 2009, pp. 371-382

Received 21/03/09
Accepted 27/04/09
Final version 08/05/09

way a human would do it, exploiting the heuristic ability. A FLC let the engineers to integrate human reasoning in computing systems.

Besides, FPGA-based design let us create, probe and modify easily and quickly any quantity of digital systems and implement dedicated systems that helps to speed up other non-real time systems.

From this point of view, it has been developed a large quantity of FLC architectures, derived from Computing Architecture. These architectures are classified by its processing way. There are sequential, combinatorial [9], parallel, pipelined and mixed models. Everyone has its own advantages and disadvantages and designers must choose which processing fits their design needs. Some designers used these architectures with other techniques that help to improve the general performance in complex algorithms. Arithmetic operations like multiplication and division are very expensive from the point of view of timing and used resources in a reconfigurable device; that is why these arithmetic operations represent a dare for designers. Some designers prefer to implement these operations to calculate a parameter of the FLC every time it is necessary [8]; this technique is called Runtime Computation (RTC). But some designs use extern elements like memories, sometimes called Look Up Tables (LUT), to calculate FLC parameters by anticipation; this another technique is called Look Up Computation (LUC) and represents a good way to improve the timing; some of this systems use supervising computing to set up the FLC configuration on line and recalculate new parameters [5:7].

However, these two techniques have some advantages and disadvantages too: LUC approach has the advantage of improving the timing critically, but has the disadvantage of using lots of external memory and a complicate way of recalculating and actualizing all the parameters of the FLC. By contrast, RTC has the disadvantage of being expensive when implementing the design on a reconfigurable device, because all the arithmetic calculation is made on line, and they need to use lots of resources on the reprogrammable device; but this represents an advantage, because it is not needed to actualize an entire LUT.

It is a dare to play with these architectures and techniques to make a balanced FLC design, by which it is necessary to change the way of designing algorithms to describe a FLC. Then, the feasibility of implementing a FLC on FPGA depends on the choice of an optimum algorithm that spend low time processing and the least quantity of used resources of the device as possible, without increasing the complexity in order to make a suitable upgrading.

This paper shows a practical approach of FLC combinatorial architecture in order to make simple construction modules and easy upgrading using a reprogrammable device, FPGA.

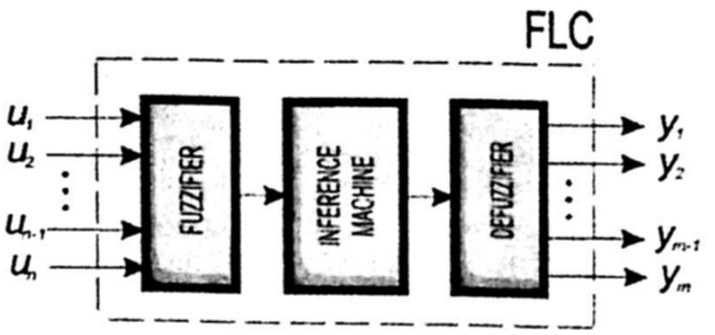


**Fig. 1.** Fuzzy Logic Controller (FLC). Assume as the system inputs and as the system outputs. There are inputs per outputs

## 2 System Description

The purpose of this section is to show a practical way of designing efficient FLC, which consists of eight steps and let us implement it easily on FPGA. Assume as the inputs to the FLC and as the outputs. Fig. 1 shows a FLC which consists of three basic stages: Fuzzification, Inference Machine and Defuzzification. The *Fuzzification* stage consists of a set of fuzzifiers, one per input that converts every crisp input into several fuzzy values or membership values. The *Inference Machine* contains the behaviour of the FLC and it is built with MIN- MAX modules; these modules are interconnected according to the fuzzy rule set inspired by one expert and decide which action will be taken based on the fuzzy values obtained from the fuzzification stage. These rules have simple inferences of the type IF- THEN. Also, the *Defuzzification* converts these inferred values onto crisp values, by means of statistical calculations, which represents the control action over the actuator.

The advantages of hardware design in FPGA, using HDL let us build any system in short time and design or redesign when needed. Next steps are required for build a FLC:

1. Establish whatever the designer want to control and which variables will be related to get it.
2. Define the number of inputs and outputs of the FLC based on the last step.
3. Define the number of membership functions or fuzzy sets for each input and output based on the last step and define their shape based on the process characteristics and operation range of the FLC (discourse universe).
4. Set the FLC configuration by means of the fuzzy inference rules according to the wished operation and based on the expert knowledge about the process.
5. Build the fuzzifier with simple membership functions simply by replication (trapezoidal, triangular, S, Z).
6. Build the inference machine based on step 4, by means of MIN- MAX modules using the building steps shown in Sect. 2.2.
7. Once inference machine is ready, build the defuzzification stage by means of multiplication and division modules using parallelism.

8. Finally, FLC can be implemented on FPGA.

These steps are related to a combinatorial architecture divided in several modules according to three RTC basic stages for a FLC, which will be described in the next subsections.

For the FLC implementation it was used VHDL, Xilinx ISE 6.3i, Mentor Graphics Modelsim Xilinx Edition III 6.0a. It is used Xilinx Spartan 3 XC3S200–5FT256 FPGA Starter Kit. In order to verify the FLC performance, it is necessary to make a simulation using the Fuzzy Toolbox of MATLAB and build a control system with SIMULINK.

## 2.1 Fuzzification

The fuzzification stage comprises a set of fuzzifiers attached to every input variable; each one parallel from the others and their performance does not depend on the others either.

We assume that all membership functions shape will be triangular, trapezoidal, S and Z, because they are the easiest to implement in hardware. Regardless the shape, the fuzzifier processing is based on the isosceles triangular shape, which comprises of several circuits like comparators (<, > and =), MUX, adders, subtracts, a multiplier and a divider, each one with a size of  bits, as shown in the Fig. 2. This VHDL module converts a crisp digital value into a membership digital value, which it is supposed to be previously converted by an ADC way, according to the input parameters: the CENTER and the APERTURE of the triangle. These two parameters of the membership functions accomplish the RTC technique in order to make the online adaptation and the FLC tuning. Once it is known the aperture it is possible to calculate the triangle slope using a divider module. Parallel with this process, the module calculates the exact position in the triangle based on the input parameter and the aperture; because of being an isosceles triangle it is necessary to know which region of the triangle the input will address (left to the center or right to the center) and the MUX drives this result to the multiplication module. Whenever the slope is ready, it is multiplied by the MUX selection and then it is obtained the membership value of this fuzzy set. The last MUX drives the results by means of the input value parameter and the center of the triangle.

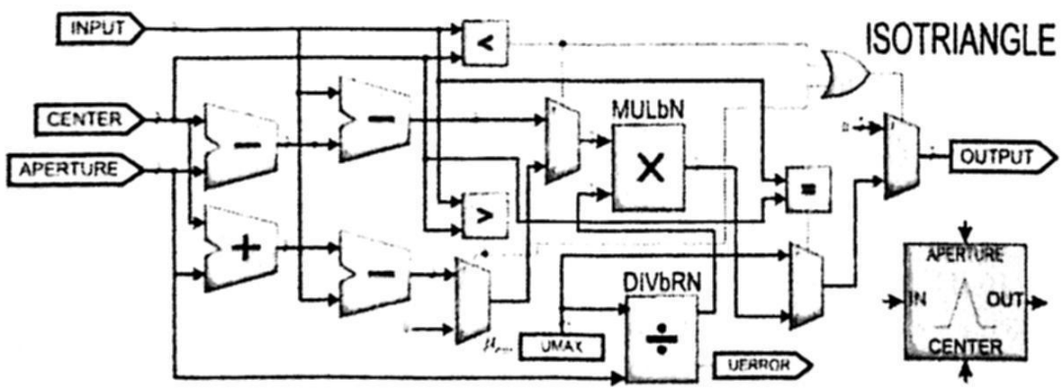


**Fig. 2.** Isosceles triangular membership function shape. This module uses combinatorial arithmetic operations like multiplication and division.

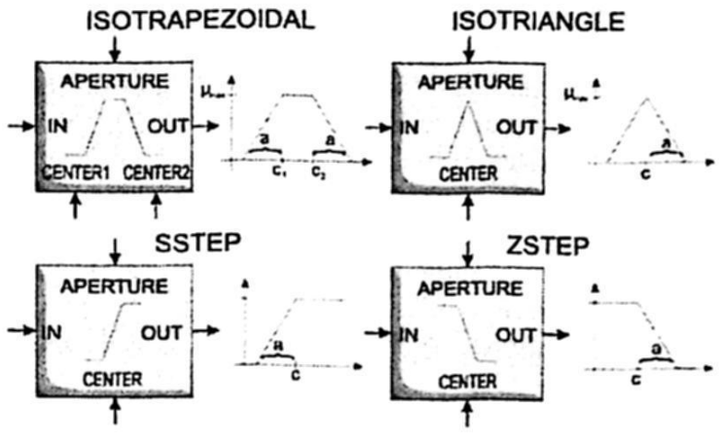


**Fig. 3.** Several hardware suitable membership functions: symmetric triangle and trapezoid and S and Z functions.

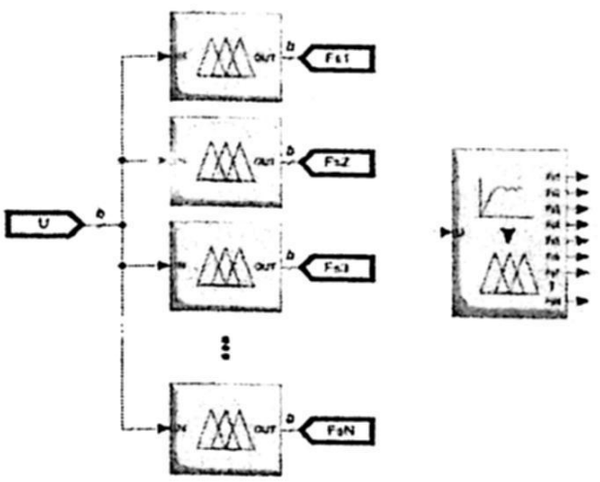


**Fig. 4.** Fuzzifier for a single input. Note that every fuzzy set is tied to the same input.

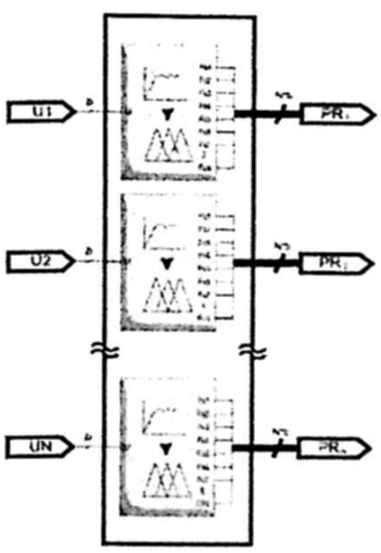


**Fig. 5.** Fuzzification Stage comprises with several fuzzifiers per input variable

The critical calculation in this module is the slope because uses arithmetic division operation. This combinatorial circuit calculate the slope whenever the aperture is changed; otherwise it is not calculated and the module uses the last calculated slope. Once the slope is known, the next critical calculation is the multiplication, and this operation represents now the bottleneck of this module (only when aperture has not any change).

Trapezoidal, S and Z membership function shapes have similar components. As mentioned before, a fuzzifier is compounded of several fuzzy sets described for membership functions, interconnected in parallel having the same input values. These functions may have several shapes as shown in Fig. 3 and the interconnection seems like it follows in Fig. 4.

Every fuzzifier has one input and several outputs, depending of the choice of the designer. Also, every input should have a fuzzifier and the interconnection seem like in Fig. 5.

The results of every fuzzifier represent the premises of the inference rule set of the FLC. Next section describes the inference machine construction according to a set of steps using Mamdani operation.

## 2.2 Inference Machine

Let us define a *premise* as the input data involved with the control, it means that an involved input will be considered to decide which control action will be taken. A *consequence* is a result of the inference, the output data of inference machine, it means the decision that FLC will take based on the premises.

A fuzzy rule set is the FLC configuration of the simple form

**IF** *premise* 1 **AND** *premise* 2 **AND**, ..., **AND** *premise n*
**THEN** *conseq* 1 **AND** *conseq* 2 **AND**, ..., **AND** *conseq m*

For instance: *"IF water is cold AND dirtiness is worst AND charge is heavy THEN washing is hard AND time is long."*

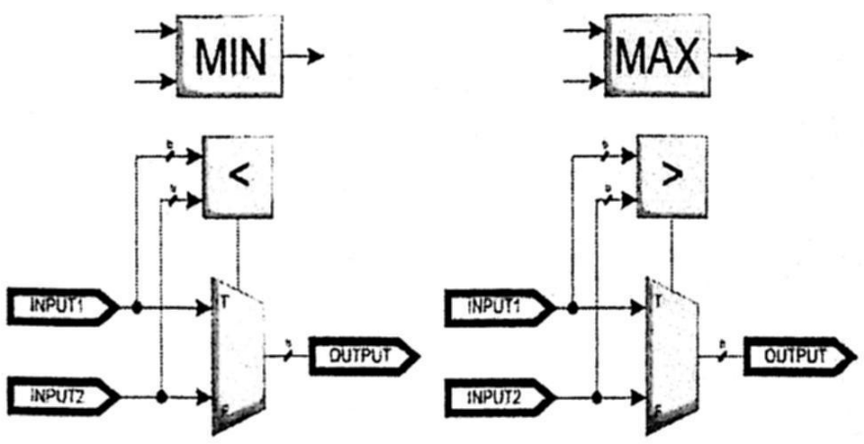


**Fig. 6.** MIN- MAX modules consist of simple MUXes with comparator circuits.

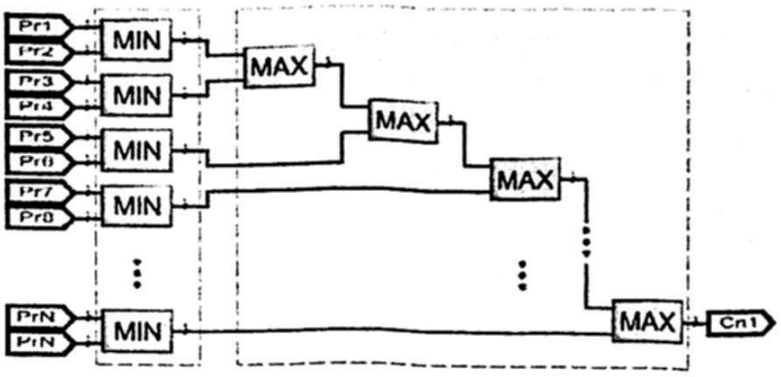


**Fig. 7.** Inference machine stage construction. All premises collaborate with a consequence.

A Mamdani inference machine consists of MAX- MIN (Fig. 6) modules interconnected according to the fuzzy rule set. This is the main part of a FLC because represents the FLC configuration [2]. Every module is connected the way the rule says and it is necessary to consider some specifications:

a) Identify all related rules that have common consequences within the fuzzy inference rule set. This common consequence rule set is a new subset.
b) Within the common subset obtained from last step, connect all related premises, according to the rule, pair by pair to MIN modules.
c) From all results after the MIN modules, it is necessary to find the maxim value of them using cascade MAX operations. This is the corresponding value of the consequence.
d) Repeat all steps for every consequence.

A MAX- MIN structure of an inference machine has MIN modules in parallel. Unlike the MAX modules are in cascade, as shown in Fig. 7.

## 2.3 Defuzzification

This defuzzification obtains a crisp output by means of output fuzzy sets, sometimes called *Centroid* method. The calculation of the centroid is made using the membership values , obtained from the inference engine, and the output fuzzy set centers [1]. It is often considered as *singleton* membership function, because of its computational simplicity and because this statistical calculation is independent of the output fuzzy set shapes.

$$y_q^{crisp} = \frac{\sum_{i=1}^{n} b_i^q \mu_i(x_1, x_2, \ldots, x_n)}{\sum_{i=1}^{n} \mu_i(x_1, x_2, \ldots, x_n)} \qquad (1)$$

This defuzzifier needs a division calculation, as seen in the Eq. 1, which results computationally very expensive. It is needed to emphasize that a multiplication results in a number of bits, which is not practical neither cheap computationally. In order to avoid the multiplication before the division, so part of Eq. 1 was implemented this way:

$$\sigma_C = \frac{b_i^c}{\sum_{i=1}^{n} \mu_i(x_1, x_2, \ldots, x_n)} \tag{2}$$

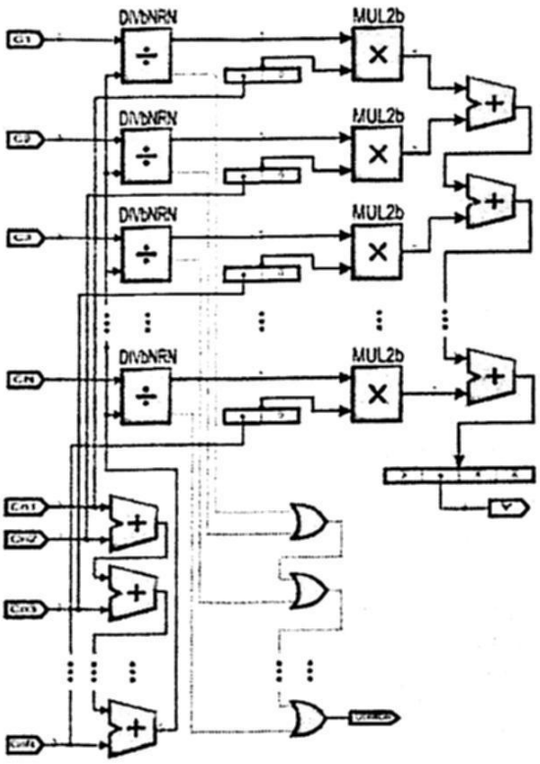


**Fig. 8.** Defuzzification stage: Centroid. This module performs several division and multiplication operations simultaneously in order to obtain the crisp output.

Where every center is divided by the summation of the membership values obtained from the inference machine. Then, the result (Eq. 2) is multiplied by every membership value obtained from the inference machine. To get this, it was needed to implement a combinatorial non- restoring division [3] modified to obtain a fixed point bits quotient, because $\sigma_i <$, as shown in Fig. 8.

The defuzzifier has two disadvantages: a) uses one division operation per consequence, which results expensive whenever the number of output fuzzy sets increases; b) though it still manages a multiplication after the division, it does not use a division which is more expensive than a multiplication.

Finally, the result is allocated in the third and second octets of the summation result (right end is LSB), integer and fractional corresponding parts.

## 3 Implementation and Verification

As example of application, a FLC for a DC servo is implemented, as mentioned above, in order to verify the correct performance of the FLC. Now, requirements of the system are explained.

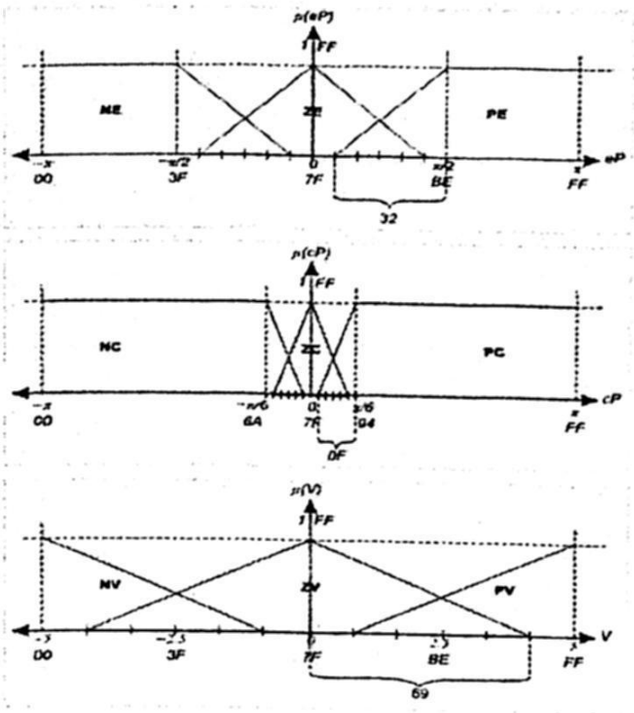


**Fig. 9.** Initial membership function distribution for DC servo. Initially, distribution of fuzzy sets is arbitrary. It can be modified later.

The control system was built in MATLAB Fuzzy Toolbox first, creating a fuzzy inference system by software (FIS). Then it is chosen a set of representative values or test bench in Table 1 and it is applied to the FIS in order to prove the FLC performance once it is implemented in the Xilinx FPGA kit. There are two versions of the FLC: the first one, named FLC1, has a specific initial set of input membership function parameters (center and aperture) and the second one, named FLC2, has been modified their input parameters in order to observe the influence of online adaptation. Finally every value in the table was compared against the FPGA results. Every obtained result is reflected in Table 1.

Suppose a 2X1 fuzzy system for a DC servo, which uses nine rules because it has three fuzzy sets per input (position error <rads>- eP: NE, ZE, PE. position error change velocity <rads>- cP: NC, ZC, PC) and output (voltage <volts>- V: NV, ZV, PV), shown in the Fig. 9, which are the following:

```
IF eP  is NE AND cP  is NC THEN V  is NV
IF eP  is NE AND cP  is ZC THEN V  is NV
IF eP  is NE AND cP  is PC THEN V  is NV
IF eP  is ZE AND cP  is NC THEN V  is NV
IF eP  is ZE AND cP  is ZC THEN V  is ZV
IF eP  is ZE AND cP  is PC THEN V  is PV
IF eP  is PE AND cP  is NC THEN V  is PV
IF eP  is PE AND cP  is ZC THEN V  is PV
IF eP  is PE AND cP  is PC THEN V  is PV
```

Then, it is provided a test bench which consists of 23 values and describes several situations.

**Table 1.** Test bench for a DC servo control.

| Case | eP | cP | V | | |
|---|---|---|---|---|---|
| | | | | I | |
| 1 | $-\frac{\pi}{2}$ | $-\frac{\pi}{6}$ | −5.89 | −5 | −5 |
| 2 | $-\frac{\pi}{2}$ | 0 | −5.89 | −5 | −5 |
| 3 | $-\frac{\pi}{2}$ | $\frac{\pi}{6}$ | −5.89 | −5 | −5 |
| 4 | 0 | $-\frac{\pi}{6}$ | −5.89 | −5 | −5 |
| 5 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | $\frac{\pi}{6}$ | 5.89 | 5 | 5 |
| 7 | $\frac{\pi}{2}$ | $-\frac{\pi}{6}$ | 5.89 | 5 | 5 |
| 8 | $\frac{\pi}{2}$ | 0 | 5.89 | 5 | 5 |
| 9 | $\frac{\pi}{2}$ | $\frac{\pi}{6}$ | 5.89 | 5 | 5 |
| 10 | $-\frac{\pi}{4}$ | $-\frac{\pi}{12}$ | −2.5 | −2.5 | −2.4 |
| 11 | $-\frac{\pi}{4}$ | $\frac{\pi}{12}$ | 0 | −0.4 | 0 |
| 12 | $\frac{\pi}{4}$ | $-\frac{\pi}{12}$ | 0 | 0.3 | 0 |
| 13 | $\frac{\pi}{4}$ | $\frac{\pi}{12}$ | 2.5 | 2.5 | 2.4 |
| 14 | $-\frac{\pi}{4}$ | 0 | −2.5 | −2.4 | −2.4 |
| 15 | 0 | $-\frac{\pi}{12}$ | −2.5 | −2.3 | −2.4 |
| 16 | $\frac{\pi}{4}$ | 0 | 2.5 | 2.4 | 2.4 |
| 17 | 0 | $\frac{\pi}{12}$ | 2.5 | 2.2 | 2.4 |
| 18 | $-\frac{\pi}{4}$ | $-\frac{\pi}{6}$ | −5.53 | −5 | −5 |
| 19 | $-\frac{\pi}{4}$ | $\frac{\pi}{6}$ | 0.59 | 0.2 | 0.2 |
| 20 | $\frac{\pi}{4}$ | $-\frac{\pi}{6}$ | −0.59 | −0.2 | −0.2 |
| 21 | $\frac{\pi}{4}$ | $\frac{\pi}{6}$ | 5.53 | 5 | 5 |
| 22 | $-\frac{\pi}{2}$ | $-\frac{\pi}{12}$ | −5.53 | −5 | −5 |
| 23 | $-\frac{\pi}{2}$ | $\frac{\pi}{12}$ | −5.53 | −5 | −5 |

Cases 1 to 9 represent those circumstances where input data is not member of a pair of membership functions at same time (no overlapping region) for every input, it means, there are 2 active rules at least. Cases 10 to 13 comprises all those circumstances where exist overlapping in every input, it means, there are 4 active rules. Cases 14 to 17 comprise all those circumstances, where there is overlapping in one variable and there is not in the other; and vice versa. Cases 18 to 23 comprises all those circumstances where the input data pertains only to one left or right end membership function in one variable and there is overlapping in the other.

As shown in Table 1, it is possible to observe the influence of overlapping, which may vary the FLC accuracy due to division truncation in defuzzification stage, which may be corrected simply by using both third octet and the second eight bits as the fractional part, as shown in Fig. 8.

Also, FLC tuning was made changing the membership function parameters of inputs and outputs. The results in Table 1 show how accuracy increased, expanding the aperture of the membership functions and the position of fuzzy singletons, as you can see in field FLC2 compared with FLC1.

Finally, Table 2 shows all timing and resources in FPGA used for every implemented module built for DC servo FLC example. DC servo FLC needs 84 ns to make an inference. Then, its processing data rate is 11.9 MFLIPS. Upgrading this architecture does not affect considerably the FLC performance, because of the parallelism of this architecture.

## 4 Conclusions

It was designed an FLC architecture using RTC combinatorial arithmetic modules. In order to get this, it was supplied to designer a practical approach for FLC design, using a study case (DC servo). Those developed VHDL modules were implemented in FPGA and it was possible to verify the FLC performance compared with the FIS simulated with MATLAB. Then, we got a FLC architecture that competes with recent developments and give us a practical fast FLC prototyping.

**Table 2.** FPGA timing and resource results obtained for DC servo control.

| Algorithm | Delay (ns) | LUT |
|---|---|---|
| 16 bits non-restoring division | 48.50 | 644 |
| Modified 8 bits non-restoring division | 28.83 | 208 |
| 8 bits restoring division | 28.84 | 124 |
| 8 bits multiplication | 13.17 | 36 |
| Isosceles triangle MF | 36.70<br>14.51 | 251 |
| S-step MF | 36.70 | 249 |
| Z-step MF | 36.70 | 251 |
| Fuzzifier | 37.42 | 755 |
| Defuzzifier | 41.49 | 677 |
| Mamdani inference machine | 19.32 | 242 |
| MIN-MAX operations | 9.36 | 16 |
| FLC | 84.01 | 2689 |

Parallelism that exists between the FLC modules improved its general performance. So, this architecture has the capability of grow modularly, adapting to the designer needs without presenting high complexity when upgrading the system. This modularity may be approached using a FIS to VHD language interpreter that simply generates the proper HDL program, using the basic modules presented in this paper, regardless the used technology, based on the MATLAB `*.fis` configuration file.

It was demonstrated that this FLC has adaptation capabilities by having online modifiable registers, being their membership function parameters which provides accuracy. Also, it was demonstrated that combinatorial design is a feasible and practical way for FLC design, obtaining results which are comparable with those reported, approaching the goods of recent FPGA technology.

## References

1. Téllez, A. Fuzzy Logic Controller Architecture using Combinatorial Logic, Instituto Politécnico Nacional. Centro de Investigación en Computación. Mexico City, 2008.
2. Patyra, M. J.; Mlynek, D.M.; "Fuzzy logic: implementation and applications;" Wiley; 1996.
3. Oberman, S. F.; Flynn, M. J.; "Division Algorithms and Implementations;" IEEE Transactions on Computers; Aug 1997; Vol 46, No. 8; pp. 833–854.
4. Togai M.; Watanabe H.; "Expert system on a chip: An engine for real-time approximate reasoning;" IEEE Expert Syst. Mag., 1986, pp. 55–62, Volume 1.
5. Vasantha Rani, S.P.J.; Kanagasabapathy, P.; Sathish Kumar, A.; "Digital Fuzzy Logic Controller using VHDL;" INDICON, 2005 Annual IEEE, 11–13 December 2005, pp. 463–466.
6. Singh, S.; Rattan, K.S.; "Implementation of a fuzzy logic controller on an FPGA using VHDL;" Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American 24–26 July 2003, pp. 110–115.
7. Deliparaschos, K.M.; Nenedakis, F.I.; Tzafestas, S.G.; "A fast digital fuzzy logic controller: FPGA design and implementation;" Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference, 19–22 September 2005, Volume 1.
8. Gaona, A.; Olea, D.; Melgarejo, M.; "Sequential Fuzzy Inference System Based on Distributed Arithmetic;" Computational Intelligence for Measurement Systems and Applications, 2003. CIMSA '03. 2003 IEEE International Symposium, 29–31 July 2003, pp. 125–129.
9. Manzoul, M.A.; Jayabharathi, D.; *"Fuzzy Controller on FPGA Chip;"* Fuzzy Systems, 1992., IEEE International Conference, 8–12 March 1992, pp. 1309–1316.