Proceedings of the 2007 IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007)

FPGA as a Tool for Implementing Non-fixed Structure Fuzzy Logic Controllers

Jose Luis González, Oscar Castillo, and Luis T. Aguilar

Abstract— **This paper presents an open architecture design to implement Fuzzy Logic Controllers (FLC) on a Field Programmable Gate Array (FPGA) integrated circuit. This non-fixed structures design is capable of Fuzzy Inference Engine (FIE) parameters on-line user specification, achieves a design space that includes Mamdani FIE, max-min rule evaluation, and weighted average defuzzification method, with user defined Membership Function(MF)type and number, MF parameters, antecedent and consequent rule construction, and output variable MFs; achieving it within a single chip, high speed, low cost, small footprint FPGA. VHDL language is use for hardware design description of a tested 8-bit resolution FLC; portability and scalability is possible with minor modifications. Algorithms, timing diagrams and hardware resources required are presented; truncation related errors and processing speed achieved is reported.**

I. Introduction

Fuzzy Logic Controllers have the ability to infer good results on conditions of imprecise and noisy data, allowing it to tackle problems where the analytic plant model is incomplete or is time-variant, situation that makes it difficult or impractical to use classical analytical design methodologies [9].

More reliable Fuzzy Logic theoretic foundations have been able to provide a variety of algorithms to implement Fuzzy Inference Engines (FIE) for control purposes, each of this with different degree of numerical computation complexity and linguistic knowledge representation.

Most FIE models have similar capabilities but it should be mentioned that cost, size (footprint), processing speed, theoretic fidelity, and flexibility must be considered to achieve a reliable working FLC. These elements are mutually exclusive then it is common to find literature where on any given working application one consideration is given priority over others (see *e.g.*, [1], [2], [6], [7], [10], [11]).

A. Motivation

Common platforms for FLC algorithm implementation are: Personal Computers (PC), microprocessor or microcontroller based slim systems, custom digital integrated circuits and custom analog integrated circuits; each of these platforms offers different levels of advantages over the others either because cost, size (footprint), processing speed, ease of implementation, theoretic fidelity, and versatility.

J.L Gonzalez is with Facultad de Ciencias Quimicas en Ingenieria de la Universidad Autonoma de Baja California (email: joseg@uabc.mx)

O. Castillo is with Instituto Tecnologico de Tijuana, Departamento en Ciencias de la Computacion (email: ocastillo@tectijuana.mx)

Luis T. Aguilar is with Instituto Politécnico Nacional, Centro de Investigación y Desarrollo de Tecnología Digital, Ave. del parque 1310 Mesa de Otay, Tijuana Mexico 22510; (e-mail: luis.aguilar@ieee.org).

Although PCs are the most versatile and offers best theoretic fidelity, due to cost, size and processing speed are not best choice for some applications.

On recent years, Field Programmable Gate Arrays (FPGAs) have been used to produce working single chip FIEs (see *e.g.*, [6], [8]); low cost, relative short design cycles and ease of synthesis are desirable characteristics that allows FPGA to be used for custom dedicated hardware designs, but only on projects like [4] that the architecture is flexible enough to reach a wide range of applications (design space). Like in ([1], [10], [6]) once the design has been chosen and synthesized on the FPGA, it can not be edited. On large design space projects ([6], [8]) that use Look Up Tables (LUT) instead of general numerical computation its versatility is restricted to a particular FIE model because LUTs are non-reconfigurable once they have been hardware synthesized, making it a fixed custom solution hardware.

A flexible, open architecture on-line programmable hardware design can broaden FLC applications if a slim, fast, low cost system be available, for this FPGA was selected for a non-fixed FLC design.

B. Objectives

Development of the architecture presented on this paper sets out to facilitate synthesis of FLC of non-fixed structures on single chip, low cost FPGA; concurrent hardware processing design achieves high speed throughput; included complementary hardware within the FPGA eases interface to a command unit, and at the same time allows for on-line FIE parameter editing and thus produce a non-fixed FLC dedicated hardware.

Use of programmable memory instead of LUTs to allocate parameters for use on general purpose numerical computation units, gives the user chance to change FIE parameters on-line and substantially increase hardware versatility, and set it on a coprocessor like category: a Fuzzy Logic Coprocessor (FLCp).

This editing capability gives FLC ability to change its control decision surface on-line, and thus makes it useful for adaptive FLC research, for which this architecture was develop.

C. Methodology

To achieve desired goals, all FLC components and complementary interface hardware is fitted within a low cost single chip FPGA IC; additionally, this FLC design employs concurrent processing whenever possible to maximize throughout.

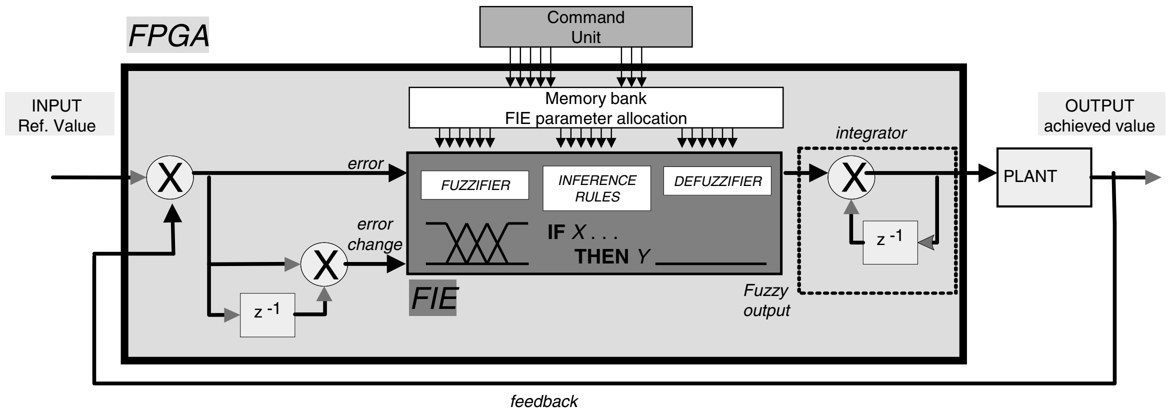

Fig. 1. Fuzzy Logic Controller general structure; FIE, complementary controller hardware and programmable memory bank.

A large design space is possible incorporating memory allocations for FIE parameters to be programmed via a command unit interface by a microprocessor or other similar device, this reduces design-to-implementation complexity simplifying it to standard memory write operation.

The designed architecture was coded using VHDL language for hardware description, a highly modular codification scheme was employ in order to enable incorporation of wider FLC model variations on future works; use of VHDL enables scalability and portability, and facilitates increases on resolution.

D. Outline

The paper is outlined as follows: Section II starts with description of targeted design space and all of Mamdani FIEs embedded variations targeted (input MFs, rule construction, output MF defuzz), and list required fuzzy inference and numerical computations to achieve them. Listed requirements are further discuss on Section III, to present necessary hardware to achieve its on-line programmability; flow diagrams are presented. On section IV some of the most relevant VHDL code is presented and discussed. Section V reports required hardware for each FLC component, in order to present scalability and portability potentials. Section VI reports execution times achieved, and truncation related errors due to the 8-bit working resolution used on the tried implementation, as it establishes fidelity to theoretic model. Section VII presents discussions on expected application for the proposed architecture.

II. Hardware Based FLC Requirements

Most FLC research and applications are PC based, this is because of ease of design-to-implementation, high fidelity to theoretic model, resources availability (memory, space), high end processing, flexibility, and most of all because its inherent general purpose architecture allows a wide range of FIE models to be executed; but this is possible at high cost, large footprint, and its centralized processing requires a processor fast enough to achieve a relative slower throughput.

On hardware based implementations, computation complexity is a mayor limitation; of available FIE models the most used is Sugenos because of its ease of computation, this is the reason why it was selected on [1], [2], [7], [11] and others. Sugenos FIE lacks direct linguistic knowledge representation as Mamdanis does allow, but Mamdanis is used less because it requires more elaborated computation and thus more hardware and is slower on sequential processing designs. For the FLC hardware presented here Mamdanis model was selected.

Figure 1 shows a Mamdani FIE embedded within a FLC in a FPGA; to perform control decision function, the FIE is complemented with additional hardware as suggested by [9] and [12], this includes: error calculation, error change calculation, and output integration functions.

All listed complementary hardware perform their processing function independently of the particular FIE embedded, and because of that are of static and fixed design. FIE parameters changes do not require modifications of these modular components, their design is as described on [11] and [12].

FLC versatility and on-line programmable capabilities lays within the embedded FIE. Mandanis FIE calls for three stage processing: Fuzzification, Rule inference and Defuzzification.

Mandanis models has many variations, for this paper purpose those listed on Table I were selected for the developed architecture; modular VHDL codification allows to exchange at design time some of those modules to use other Mandanis model variations like other Membership Function (MF) types, rule evaluation, or defuzzifications method.

Design space achieved is limited by variations listed on Table I. FIE data Input are error and change of error, each of these are process during Fuzzification by Membership Functions (MF), their type, quantity and position within domain are user defined. MFs types possible in this architecture are Trapezoidal and/or Triangular MF, up to 8 of them, and can be position along input doming freely; Figure 2 shows a 7 MF input partition of trapezoidal and triangular MF

TABLE I
MAMDANIS FIE TARGETED DESIGN SPACE; USER DEFINED COMPONENTS AND PARAMETERS.

Processing	Type/Method	Quantity
Input	n/a	2
Fuzzification: MF	-Trapezoidal -Triangular	2 . . . 8
Rule evaluation	-max-min	2 . . . 20
Defuzzification	-Weighted average	n/a
Output variable MFs	-Trapezoidal -Triangular	2 . . . 5

combination.

Rule evaluation is done by max-min method, and this system is capable of handling up 20 rules; antecedent and consequent rule parts are user defined. As in (1) antecedent connectives (w and y) activate on some degree one output MF (z).

$$\text{IF } x = X_0 \text{ AND } y = Y_0 \text{ THEN } z = Z_0. \tag{1}$$

On any given FIE, a maximum rule count can be of k-rules as describe by (2)

$$k = n_x \cdot x_y \cdot \ldots \cdot n_v. \tag{2}$$

where n stands for quantity of MFs on each input variable x, y and v. Given that this FIE handles two input variables with up to eight MFs each, 64 rules are possible. As proposed in [1] and [3] not all rules contribute much and are not necessary; presented system is capable of handling up to 20 user defined rules. The antecedent part of the rules require 20 comparison operations to carry out the min operation (one per rule), the consequent part of the rules require 5 comparison operations to carry out the min operation (one per each output MF).

Deffuzification is done with weighted average method as in (3), and the output variable can be partitioned with up to five MFs, of trapezoidal or triangular type.

$$z_0 = \frac{\sum_{l=1}^{n} Z_l \times \mu(z_l)}{\sum_{l=1}^{n} \mu(z_l)} \tag{3}$$

User defined capabilities requires that FIE parameters are not hardwired during design description, instead, to achieved the non-fixed structures FIE parameters are stored on memory allocations that are programmed on-line.

III. NON-FIXED STRUCTURES FLC DESIGN

Given that a high speed, dedicated FLC was desired on a dedicated hardware, and that FIE parameters were to be edited on-line, two operation modes were implemented: Program mode, and Fuzzy Inference mode. During each mode of operation, state machines controls execution of relevant dedicated modules, a system level operation description is presented on Figure 3.

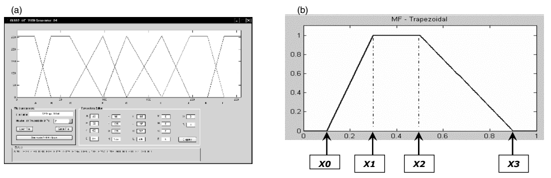

Fig. 2. State machine operation; program mode state and fuzzy inference mode state.

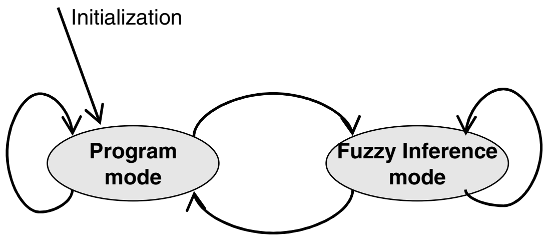

Fig. 3. Fuzzy Inference Mode State Machine; nine operation states and one wait state (for program mode operation).

As shown in Fig. 3, FLC initialization starts in Program mode, during which FIE parameters are written to memory allocations within the FPGA hardware; once the FIE to be emulated is programmed, the system is switch to Fuzzy Inference mode, were input data is process thru the FLC and outputted as control decision

Operation modes shown in Figure 3 represents each a state machine that activates relevant hardware modules. Next, Inference Mode state machine is presented, including listing of hardware modules it controls; then, non-fixed modules are discuss including flow diagrams from which VHDL code is obtained. Finally, program mode related hardware is presented.

A. Fuzzy Inference Mode State Machine

During Fuzzy Inference mode of operation, dedicated hardware that carries out all process related to the FLC operation, including the 3-stage Mandani FIE algorithm are activated, and data is passed sequentially until output is achieved, new data is then inputted to repeat the process.

Figure 4 shows the state machine for the Fuzzy Inference mode operation; it is a 10 state machine that activates hardware modules described on Figure 1 as follows:

St0_Wait: A wait state to pause Fuzzy Inference operations while on Program Mode, during which new FIE parameters are written to memory by the command unit interface.

St1_DataRead: During this state, external input data is uploaded, consist of conventional asynchronous write cycle.

St2_Error: This state activates subtraction between Reference Value and Achieved Value, thus, it calculate error input variable.

St3_ErrChange: During this state a subtraction between present and previous error is made to calculate error change

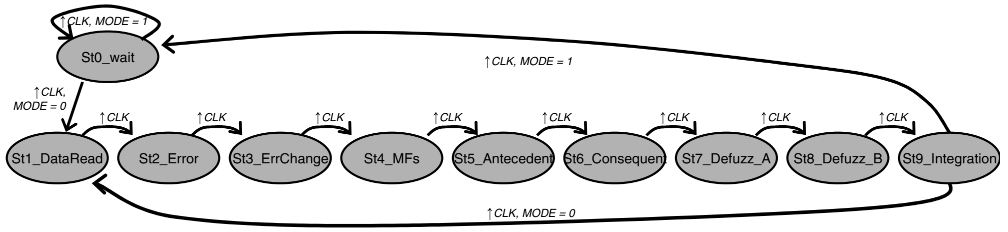

Fig. 4. Fuzzy Inference Mode State Machine; nine operation states and one wait state (for program mode operation).

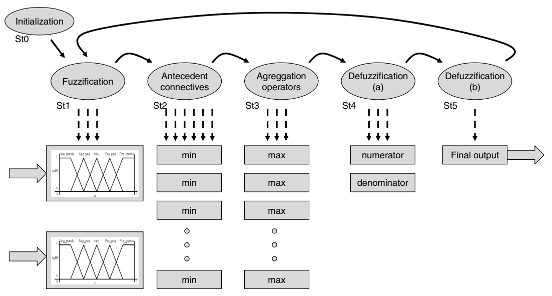

Fig. 5. Non-fixed structures related states; dedicated hardware activated during each state.

input variable.

St4_MFs: This state activates the two fuzzification modules, and their eight MF each. Memory stored data is used as fuzzification parameters.

St5_Antecedent: This state executes the 20 min operator modules as the antecedent connective operators. Memory stored data is used with this modules to construct rules antecedents.

St6_Consequent: During this state, five max operator modules are activated as the consequent aggregation operators. Memory stored data is used with this module to construct rule consequents.

St7_DefuzzA: This state activates two defuzzification modules as detail on next sections. Memory stored data is used to incorporate user defined output MFs center values.

St8_DefuzzB: This state activates third defuzzification module that calculate final inference result.

St9_Integration: During this state inference result is integrated to calculate final control decision output.

Fixed structure hardware modules are those with process that are independent of the FIE implemented, this structures are activated during St1_DataRead, St2_Error, St3_ErrChange, St8_DefuzzB, and St9_Integration. These structures are conventional digital arrangements as describe on [11] and [12], and their details not included on this paper.

Figure 5 shows states four thru seven and the respective non-fixed structures that each one activates.

To meet high speed throughput and as Figure 5 shows with vertical dashed arrows, each state activates simultaneously as many dedicated modules as needed by the required operation and FIE parameters; these quantities correspond to those listed on Table I, that is, 2 fuzzification modules (one per input variable), 20 min operator modules (one per each rule), 5 max operator modules (one per each output MF), and three modules for defuzzification process purposes.

Details for each type of modules are presented next, concurrent operation and topology arrangements are presented.

B. Fuzzification Modules

Given that the FLC must accommodate two input variables (error and error change), two independent concurrently operated dedicated modules for fuzzication operation where incorporated.

Each of these modules, as detail in [10], evaluates MFs like those shown in Figure 2(a); before degrees of membership to each linguistic variable are assign it must evaluate on which domain interval input data is in, this is done using (4) with Figure 2(b) related data.

$$\mu_A(x) = \begin{cases} 0 & x < x_0 \\ f'(x) & x_0 < x < x_1 \\ 1 & x_1 < x < x_2 \\ f''(x) & x_2 < x < x_3 \\ 0 & x_3 < x \end{cases} \tag{4}$$

where

$$\begin{aligned} x' &= x - x'_0 & (5)\\ x'' &= x_3 - x' & (6)\\ f'(x') &= m_1 x'' & (7)\\ f^{''}(x^{''}) &= m_2 x^{''}. & (8) \end{aligned}$$

Then one or two linguistic variables (those on the slope regions) may be assign degrees of membership different than '0', a numerical calculation must be executed to determine this values, this is detail on previous work [3] and [10], and done with equations (5)-(8).

For FLC user defined and on-line editing capabilities, all MF parameters values are store on memory allocations within the FPGA. Figure 6 shows data flow diagram; on it, input data is compared with parameters that define interval on all input domain.

As explain in [3], [5] and [10] a subtraction may be needed (5) (6), before a multiplication operation (7) (8) with slope data; which numerical values are used for these operations is controlled by multiplexers once the correct interval is know; final assignments are made with these arithmetic results and/or 0 value degree of membership. Multiplexers (Mux)

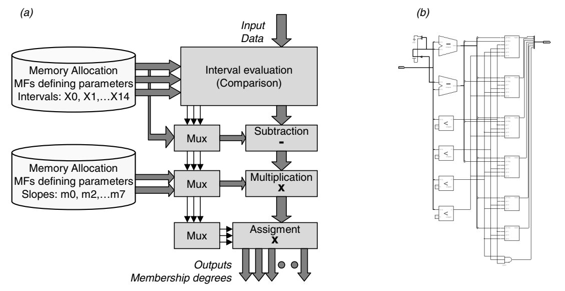
(a)
Input Data
Memory Allocation MFs defining parameters Intervals: X0, X1,...X14
Interval evaluation (Comparison)
Mux
Subtraction -
Memory Allocation MFs defining parameters Slopes: m0, m2,...m7
Mux
Multiplication x
Mux
Assigment x
Outputs Membership degrees
(b)

Fig. 6. Membership Function Modules, data flow diagram.

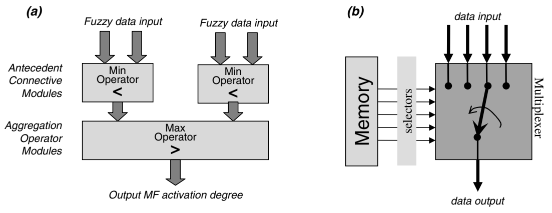

Fig. 7. Rule Base modules; (a) max-min rule inference method modules, (b) memory controlled multiplexer for data router function.

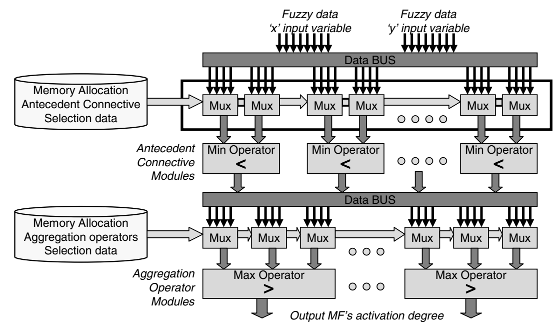
Fuzzy data 'x' input variable
Fuzzy data 'y' input variable
Data BUS
Memory Allocation Antecedent Connective Selection data
Mux
Antecedent Connective Modules
Min Operator <
Data BUS
Memory Allocation Aggregation operators Selection data
Aggregation Operator Modules
Max Operator >
Output MF's activation degree

Fig. 8. Rule base modules data flow diagram.

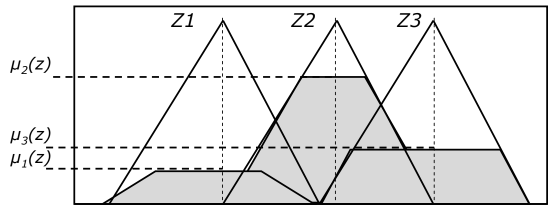

Fig. 9. Typical defuzzification data, for a 3 MF output variable example.

allow sharing one subtraction and one multiplication circuit between all MFs, as one (or maybe none) of these operation would be needed, reducing hardware requirements.

C. Rule Base Modules

Once data has been fuzzified, fuzzy rule inference is executed for each rule as (1) during states St5_Antecedent and St6_Consequent of the Fuzzy Inference Mode state machine. As previously stated, fuzzy antecedent connectives and aggregation operators were selected to be evaluated by max-min method, sequence shown on Figure 7(a).

Because rule base is to be programmed on-line, no hardwire connection was incorporated at hardware design time. To specify how antecedent connectives are to be constructed, multiplexers were incorporated to route selected MF output to antecedent connectives module inputs and, for rules consequent construction additional multiplexers were added to do the same data selection for aggregation operator functions. Figure 7(b) shows a multiplexer data selector's inputs controlled by data stored on a memory.

Figure 8 shows data flow diagram of rule base evaluation algorithm. During state St5_Antecedent fuzzy data produce by the MFs modules is available to all 40 antecedent connectives layer multiplexers, each is controlled by memory stored data. Multiplexers are paired for every one of the 20 available rules. When data reach antecedent connective modules, a comparison is made and min operator function is executed on every one of the 20 rules antecedents.

In state St6_Consequent once the antecedents are evaluated, aggregation operators must be executed on those cases were one output MF is activated by more than one rule, this is done with max operator. Aggregation operator construction employs same previous strategy, so 20 multiplexers on a consequent layer were created, each one selects one of the antecedent connective data and routers it to the max operator hardware modules.

Every four multiplexers sends data to max operator modules inputs, there are five of this modules (one per output MF), so each output MF can be activated by as much as four rules. A comparison and data selection is carried out, output is send to next state hardware modules for defuzzification operation. As figure 8 shows, independent concurrent operation of each of the layered hardware modules is possible and was specified by VHDL code.

D. Defuzzification Modules

As stated previously, weighted average method was selected as defuzzification process. The FLC was design to handle up to 5 output MFs, so (3) is expanded for this five valued possibility as (9).

$$z_0 = \frac{Z_1' \cdot \mu(z_1) + \ldots + Z_5' \cdot \mu(z_5)}{\mu(z_1) + \ldots + \mu(z_5)} \tag{9}$$

where Z_n' stands for center of each of the n-th output MFs (where $Z_n \in z$), and $\mu(z_n)$ stands for the activation degree of each one. Figure 9 shows a typical area to evaluate for a three output MF case.

Due to numerical computation complexity and to minimize hardware requirements, (9) was executed on two steps (St7_DeffuzA and St8_DefuzzB); on the first one, two dedicated modules were implemented, one to calculate numerator and another for the denominator, these execute concurrently.

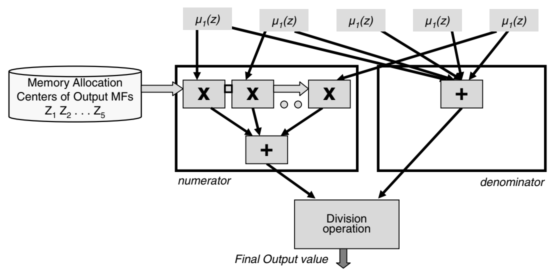

Fig. 10. Deffucification data flow diagram; user defined centers of output-MFs.

On state St8_DfuzzB, division operation was executed, but because of its relative high degree of complexity, it was implemented by a look up table strategy, because this process is independent of the FIE implement it does not limit system flexibility. Taking advantage of the fact that only integer values in the range 0 to 255 are expected (for the 8-bit target resolution design), the position of most significant bits on both numerator and denominator, and organization of expected results on this reduced search space memory requirements for this look up table are significantly reduced.

Figure 10 shows data flow for defuzzification hardware modules; on it, multiplication operation hardware modules use memory stored data as user defined parameters of centers of each output MF.

E. Program Mode Related Hardware

Command unit interface hardware is a fixed structure module, it allows communication to an external command unit as shown on Figure 1; manages data storage to memory allocated within FPG with a typical memory write cycles.

Once data is written, memory contains data use as FIE parameters on non-fixed structures FLC components. Figure 11 shows the structures for the external command unit interface hardware, this is made up of a typical address decoder circuit and a address memory array; once all values are stored FLC has been configured for a particular FIE model, and data process can proceed once operation mode is change to fuzzy inference mode.

Next, key VHDL code is presented, and portability and scalability opportunities are pointed out.

IV. VHDL System Code

For portability and scalability purposes only primitive VHDL description standard codes were employed; when ever possible concurrent process was implemented.

Data mention on previous sections falls within 3 VHDL categories: Port, Signal and Variables.

Port type data are those that correspond to actual FPGA input or output electrical signals, and are how the device communicates with external components; as such, VHDL Port declaration is as follows:

```
Port (CLK : in std_logic; --clock signal
Mode : in std_logic; --mode control
Write : in std_logi; --write enable
RDY : out std_logic; --ready signal
Ref_Value :in std_logic_vector(7 downto 0);
Real_Value:in std_logic_vector(7 downto 0);
Output : out std_logic_vector(7downto 0);
Data : std_logic_vector (7 downto 0));
```

Mode and RDY one-bit signals are for handshake purposes, DATA is an input data port thru with data is written to embedded FPGA memory for FIE parameter storage.

Ref_Value, Real Value are 8-bit data inputs for acquiring target value and actual value of physical variable; Output is an 8-bit output data port thru which fuzzy control decision is send to targeted plant or poser amplifier.

Signal type data are electrical signals that must physically exits within FPGA hardware; this signals are created by embedded modules and use local buses as Figure 1 and 4 shows to send data from one module to the next. Each hardware module on Figures 5, 6, 7, 8 and 10 are interconnected via these type of signals; activation signals send by state machines correspond too to these type. Following VHDL shows partial list of signal type data.

```
TYPE type_sreg IS (St0_Wait, St1_DataRead...
SIGNAL sreg, next_sreg : type_sreg;
SIGNAL error : std_logic_vector(7 downto 0);
SIGNAL changeErr :std_logic_vector(7 dowto 0);
...
SIGNAL ErrMF0, ErrMF1,ErrMF3,...,ErrMF7 :
       std_logic_vector(7 dowto 0);
SIGNAL ChaErrMF0, ChaErrMF1,...,ChErrMF7 :
        std_logic_vector(7 dowto 0);
...
```

`TYPE` declares data type for state machine registers (sreg) with its possible values (one per each estate); then current estate and next stare are declared.

`SIGNALS` like error carries error calculated data from producing modules to where it is required; same is done for error change signal on next line of code. Each MF values produced on Fig. 6 are send to Multiplexers on Fig. 8 thru signals as signal are created with ErrMF0 for Error MF number 0, and the rest. Same is done for data from Error Change fuzzyfication module.

Embedded memory with stored FIE parameters values, connects its data to corresponding hardware where is needed thru signal type bus declaration as showed.

Variable type data handling is declared and used within a hardware module, and used for hardware behavior description; because is not needed outside its module it may disappeared during synthesis process if it can done so by minimization and optimization algorithms employed by the FPGA development software.

VHDL reserved word `std_logic` describes one-bit long port, signal or variable; reserved word `std_logic_vector` (7 `downto` 0) describes an 8-bit long port, signal or variable.

VHDL code modification of system resolution make scalability an easy task, limited only by the amount of

TABLE II
REQUIRED RESOURCES FOR FIXED STRUCTURE COMPONENTS.

Module	logic Units Required
State Machine (10-state)	12
Error Evaluation	43
Error Change evaluation	46
Output integration	60

TABLE III
REQUIRED RESOURCES FOR MF MODULE SYNTHESIS PER TRIED RESOLUTION.

Resolution	Logic Units required	% (out of 3,840)
8-bit	304	7.9
10-bit	374	9.7
12-bit	443	11.5

electrical resources to be required. Use of only primitive VHDL instructions and avoiding custom libraries assures code portability to other larger devices, or other technologies or device manufactures.

V. FPGA System Hardware Synthesis

FLC VHDL code was synthesized for a Xilinx Inc. FPGA device type Spartan-3 XC3S200, which has 3,840 logic units, and is a low cost and offers low to midrange electrical resources.

Table III shows required logic resources used for relevant fixed structure modules, as those shown on Figure 1, a 8-bit resolution FLC was implemented with these modules.

For non-fixed structures like MFs, Multiplexers and others, resources required are related to the target resolution; denser logic was required by MFs module; tests for 8, 10 and 12 bit resolution were as presented on Table III.

As table III shows, system syntheses efficiently relative to the amount of available FPGA resources, making scalability reachable.

VI. Experimental Performance Verification

Reliable operation of any given electrical system is related to the fidelity with which emulates a target theoretic model whose throughput is to reproduce. For any digital base system, word length is of mayor concern as it determines systems fidelity to the theoretic model and as such, limits expected behavior.

Truncation errors are subjected to the 8-bits resolution targeted design; the 8-bit resolution sets the expected minimum error threshold; additional truncations errors has to be considered on all hardware modules that are required to execute operations such as multiplication and division, this occurs on MF modules, and two defuzzification modules.

On MF modules and as presented on [10], the additional error is $\pm 1/2$ value of the least significant bit (LSB), and it is due to a roundup executed during multiplication operation, this error is uniformly distributed on input domain.

On defuzzification numerator module, additional multiplications operations are executed and same roundup errors are produced, but these cancel up as up 5 of these errors are present ($\pm 1/2$ LSB), a maximum error of ± 1 LSB is observed.

Finally, defucification division module, adds one $\pm 1/2$ LSB error. Overall error is determined per FIE parameters programmed, maximum error observed on simulations is ± 1 LSB.

Throughput achieved with this FLC architecture is as follows. During fuzzy inference operation mode, each inference takes nine clock cycles. During program mode operation, all or some of the values stored on the memory array may be change (MFs, and/or rule base, and/or defuzz parameters), an overall FIE change takes 63 memory write cycles.

Execution times measured and those from simulation software gives similar results and differences may be produce by measurement instrument; typical module input-to-output delays were found to be on average 7.9 ns, and none were over 9 ns, this allows FPGA operation at 50 MHz clock speed with out errors.

VII. Conclusions

A hardware based, single chip, FPGA based, open architecture dedicated Fuzzy Logic Controller was presented; its writable memory FIE parameter control allows it to emulate a variety of Mamdani Inference Engines, including non-fixed number of membership functions on each of its 2 input variables (8 MF maximum on each input), 20 units of dedicated hardware for same number of fuzzy rules, whose antecedent construction and rules connectives are editable on-line, allowing a large design space within same hardware architecture, given this system ability to achieve high throughput processing speed, with know fidelity to theoretic model to ± 1 LSB.

VHDL codification of showed data flow diagrams permits design portability to other larger FPGA devices, and VHDL code modification can increase data resolution with only minor changes as shown. Scalability test of VHDL showed consisting implementation thru resolutions tested.

Fuzzy inference control decision at 9 clock cycles allows to use this system as a dedicated on-line editable Fuzzy Logic Controller at a low cost and with very small footprint.

Fast on-line programmable structures permit changes on the FLC for future hardware implementation of adaptive fuzzy controller, and its conventional interface subsystem allows it to be considered as a peripheral dedicated hardware on microcontroller or microprocessor based digital systems.

References

[1] F. Boshchetti, A. Gabrielli, E. Gandolfi and M. Masseti, "Digital Membership Function Generators and Non-contribute Rule Eliminator for High Speed Fuzzy Architectures", *Proceedings of the 1995 World Congress on Neural Networks.*

[2] H. Eichfeld, T. Kunenmund and M. Menke, "A 12-b General Purpose Fuzzy Logic Controller Chip", *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 4, Nov. 1996.

[3] J. L. Gonzalez Vazquez, "Analisis de Controladores Difusos Implementados en Procesadors Digitales FPGAs", M. Sc. thesis, Instituto Tecnologico de Tijuana, 2006, Tijuana, Mexico.

[4] J. L. Gonzalez-Vazquez, O. Castillo, L. T. Aguilar, “A Generic Approach to Fuzzy Logic Controller Sinthesis on FPGA”, *World Congress on Computacional Intelligence*, Vancouver Canada, 2006.

[5] J. L. Gonzalez Vazquez, O. Castilloa and L. T. Aguilar, “Sintesis de Funciones de Membresia Trapezoidales para Implementacion de Controladores de Lggica Difusa”, *HAFSA Internacional Conference on Fuzzy Systems, Neural Networks and Genetic Algorithms*, Tijuana, Mxico, 2005.

[6] V. H. Grisaldes, J.E. Bonilla and M. A. Melgarejo, “Diseño e Implementación de un Controlador Difuso Basado en FPGA”, IWS 2001, VII Workshop IBERCHIP, Motevideo, Uruguay.

[7] D. L. Hung, “Dedicated Digital Fuzzy Hardware”, Micro, IEEE, vol. 18, no. 4, pp. 31–39, Aug. 1995.

[8] E. Lago, C. J. Jimenez, D. R. Lpez, S. Sanchez Solano, A Barriga, “XFVHDL: A Tool for Sntesis o Fuzzy Logic Controllers”, *Desing Automation and Test in Europe (DATE’98)*, Paris, pp. 102–107, Feb. 1998.

[9] Timothy J. Ross, *Fuzzy Logic, With Engineering Applications, 2nd Ed.* England: John Wiley & Sons, 2004. ISBN 0-470-86075-8.

[10] S. Sanchez-Solano, A. Cabrera, C. J. Jimnez, P. Brox, I. Barutone, A. Barriga, “Implementación sobre FPGAs de Sistemas Difusos Programables”, *IWS 2003, IX Workshop IBERCHIP*, La Habana, Cuba.

[11] S.X. Yang, H. Li, Max, Q. H. Meng, P. X. Liu, “An Embedded Fuzzy Controller for a Behavior-Base Mobile Robot with Guaranteed Performance”, *IEEE Transactions on Fuzzy Systems*, vol. 12, no. 4, Aug. 2004.

[12] Li-Xin Wang, *A Course in Fuzzy Systems and Control*, Prentice Hall, U.S.A., 1996.