Complex Intell. Syst. (2016) 2:83–98
DOI 10.1007/s40747-016-0017-5

ORIGINAL ARTICLE

FPGA implementation of rule optimization for stand-alone tunable fuzzy logic controller using GA

Bhaskara Rao Jammu[1] · Pushpak Pati[2] · S. K. Patra[1] · K. K. Mahapatra[1]

Received: 20 November 2014 / Accepted: 7 May 2016 / Published online: 23 May 2016

Abstract In this paper, self-tuned, rule-optimized multi-input and multi-output (MIMO) fuzzy logic controller (FLC) is implemented on field programmable gate arrays (FPGA). The design of membership functions, rule base are made with aid of genetic algorithm (GA). Flexibility in FPGA design is implemented through tuning of FLC parameters. The system is modularized as rule base development, rule base transfer and computations on FPGA. Based on the system, an experimental dataset is obtained, which is utilized in a capable computing platform so as to develop a fine-tuned fuzzy rule base. The synthesized rule base is transferred to FPGA along with user provided inputs through a GUI. The GUI also displays the output result sent by FPGA. The communication between the GUI and the FPGA is done via universal asynchronous receiver and transmitter. Rule-optimized FLC is implemented on Xilinx Virtex-5 LX110T board. This dedicated single chip architecture performs high-speed fuzzy inferences with processing speed up to 760 KFLIPS at a clock frequency of 247 MHz using 8 rules, 2 input variables at 16-bit resolution. Experiments of software implementation and hardware software co-design implementation are presented and compared.

Keywords Fuzzy logic controller · Genetic algorithm · FPGA · UART

✉ Bhaskara Rao Jammu
j.bhaskararao@gmail.com; 511ec103@nitrkl.ac.in

Pushpak Pati
pupati@microsoft.com

S. K. Patra
skpatra@nitrkl.ac.in

K. K. Mahapatra
kkm@nitrkl.ac.in

[1] Department of ECE, National Institute of Technology, Rourkela, Orissa, India 769008

[2] Microsoft India (R&D) Pvt. Ltd., Microsoft Campus, Gachibowli, Hyderabad, India

Introduction

Fuzzy logic controllers (FLC) have immensely contributed to the industrial sector [1–4]. They have been of greater use to solve complex and non-linear control problems. Fuzzy logic, being the mathematical emulation of human reasoning, FLCs are developed on the basis of human intelligence which helps in designing intelligent control systems with advanced features in handling environmental changes and system level faults.

Fuzzy systems are designed to take certain inputs from the operator and do some operations on the taken inputs to provide desired output. It constitutes of detailed methods, procedures, routines that carry out a specific activity that are represented by the combination of different variables related through several mathematical relations and operations. The entire representation may constitute of simple or critical or both types of mathematical computations, which are difficult to synthesize practically and may give numerous errors even with best design implementations. Classical theory fails to design the systems completely and efficiently because of the increased number of variables and conditions and has become less powerful with the requirement of multiple-input–multiple-output system. To avoid such kind of problems and to design the system with easy user interfaces when input–output data are given, different mechanisms can be followed like neural networks, regression, evolutionary algorithms, fuzzy logic, etc. [5,6].

Systems are first trained with some known input–output results using any of the above methods or a combination of them and then tested on new input values. In this paper, the approach of fuzzy logic-based system design is discussed.

A number of FLCs have been designed with numerous algorithms and intelligence techniques [1–4]. But the designing of these controllers require thorough knowledge about the controlled process. FLCs are designed based on human intelligence, i.e., by the experts. This includes a chance for human error. Most of these processes are nonlinear and depend on a large number of parameters which results in the rigorous mathematical representation of the process. It is very difficult to incorporate each of the parameters while designing the FLC. Fuzzy logic can be used to solve problems such as input–output problems, classification problem and mapping problems. This paper includes methods to solve the input–output problem, i.e., the fuzzy system will be modeled approximating a system which takes some input from the user, does complex mathematical computations with it and provides output, e.g., PID controllers [7]. The actual system may have a number of problems and high level of complexity during computation. The designed fuzzy system reduces the level of complexity with a number of advantages over the original system.

Fuzzy logic controllers (FLC) are better compared to PID controllers in terms of convenient user interface, less complex mathematical modeling and fast response with virtually zero overshoot. This paper states designing of an optimized FLC using genetic algorithm (GA). It extracts tuned rule base automatically by analyzing the training data sheet only, so it is better than regular FLCs where,

1. Humans are relied to make the decision making process.
2. Enough prior knowledge is required to do the decision making.
3. Many conditions are to be checked to design the system which is not possible in the part of a person to remember and utilize properly.
4. Solution to the system must be made understandable to non-experts.

The FLC designing is an offline process, so it is tuned before it is used. Among the purported methods are the following: Nomura [8] reported a self-tuning method for fuzzy inference rules employing a descent method for TS fuzzy rules with constant outputs and isosceles triangular MFs. Glorennec [9] presented an adaptive controller using fuzzy logic and connectionist methods. Siarry and Guely [10] used the gradient descendant method for optimizing TS rules with symmetric and asymmetric triangular MFs and output functions, proposing the ‘centered TS rules’ for avoiding a specific class of local minima. Cordon and Herrera [11] used real-coded GA with some genetic operators for tuning the membership points. On the other hand, some approaches using GAs for designing an adaptive FLC have been presented in the literature. The facility of FLCs to capture the automatic learning from data and render it into a rich control strategy is applied in these FLC’s, without the need of mathematical model of the system or expert knowledge. The mathematical model of the system under control has led to a significant increase in the number of control applications in the last fifteen years [5,12]. This has propelled the development of different approaches to implement fuzzy inference systems. These approaches range from completely software or hardware solutions. This paper goes to hybrid realization which allows adequate tradeoff between flexibility and inference speed [13,14]. Hybrid strategies require software task execution and fixed hardware to execute complex time-consuming tasks usually the Fuzzy inference process (FIP) [15].

Field programmable gate arrays have a consistent track record of capability growth, and they track Moore’s law better than any other semiconductor device. As an Example, XC2064 FPGA, introduced by Xilinx in 1985, had 1024 logic gates. The largest, Xilinx Virtex-7 XC7V2000T, has 15,636,480 gates, an increase of 10,000 times over the past 26 years. If the current trend of capacity and speed increase continues, FPGAs will essentially become highly integrated SoC platforms which include multi-core processors, wide range of peripherals, and high-capacity logic fabric, all in one chip.

Software level designing has the upper hand to hardware level designing in terms of validating several conditions, handling add-in functionality for the system and easier configurability, where as hardware is faster and functionally superior to software in terms of processing high volume of data for computation. Software can be embedded in memory circuits, so as to come up with a hardware–software co-design. Through this type of design principle, the overall system will be having the high-performance capability of hardware as well as the smooth designing and reprogrammable capability of software. This co-design has been made possible with the development of high-performance processor core, embedded memory circuits, faster communication technologies between hardware & software.

This paper used the hardware/software co-design for the fuzzy system designing and direct interaction between user interface (PC) and the FPGA has been done over serial communication through UART.

The brief description of the paper is as follows. The next section briefly describes the basics of the entire system architecture followed by which the method for rule base extraction is proposed. The subsequent section describes the

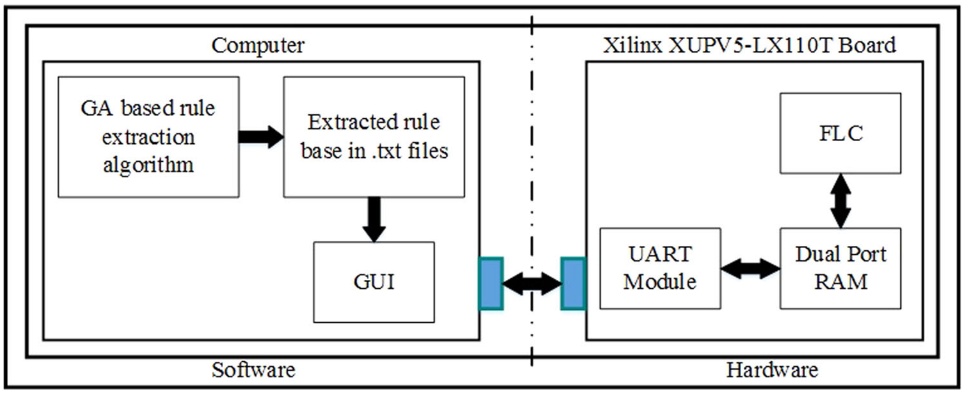

Computer
Xilinx XUPV5-LX110T Board
GA based rule extraction algorithm
Extracted rule base in .txt files
GUI
FLC
UART Module
Dual Port RAM
Software
Hardware

Fig. 1 System architecture of GA-FLC on FPGA

method for rule base and input values transmission to the FPGA board. Then the hardware architecture of FLC and its design constraints are presented. Before the concluding section, experimental results of the suggested architecture to the existing methods for input–output problems such as Hang function, chaotic time series and their hardware implementation results are explained. Finally main conclusions are summarized.

System architecture

The proposed system architecture for GA-based FLC on FPGA using software–hardware co-simulation is presented in Fig. 1. The training data sheet prepared on the experimental observations of the system is provided to the computer. The training data sheet is used to extract the initial rule base and then optimize it using GA. Accuracy of the rule base depends on the distribution of data points in the data sheet over the range of application.

Once the rule base has designed the parameters of the rules, input variables and control information are sent through serial communication protocol like RS-232 of the computer to the UART module of the Xilinx board and saved in the dual-port block RAM on the FPGA through its Port A. This completes the updating of the rule base at the hardware level. Then for testing purpose input parameters of a data point are taken through a GUI on the computer and sent serially to the Xilinx board. All the rules, input values, control information are read from port B of the dual-port RAM, which initiates the fuzzy inference processing to get the desired crisp output. This value is again transferred serially back to the computer for display on the GUI. The memory address map and control information of dual-port RAM are shown in Table 1. All the parameters are notated in fixed point notation Q8.8 for simplicity in hardware implementation. Since UART sends data in bytes, each data consume two 8 bit addresses in RAM.

Here the chore is divided into two sections, i.e., software section and hardware section. The software part is done in the personnel computer (PC) for the designing and tuning of the rule base and the hardware section includes real-time implementation of the FLC on the XUPV5LX110T board. The specification of the FLC proposed to be constructed in this paper is as follows:

1. No of inputs: 4.
2. No of outputs: 2.
3. Shape of membership function: triangle.
4. Number of fuzzy sets per input and output variables: 7.
5. Resolution of membership values: 16 bit.
6. Implication model: Mamdani.
7. Aggregation model: Mamdani.
8. No of rules are field programmable.
9. Configurables on field by GUI application.

Table 1 Memory space

Address	Access	Name	Description
00H-01H	Read/write	CTRLREG	Control register for software and hardware updation
01H-221H	Read	RULEBASE	Based on the number of rules the address range is varied within this address with 16 bit resolution
222H-225H	Read/write	INPUTVAL	Address to store input variables in 16 bits
226H-22AH	Write	CRISPOUT	To store final crisp output

The entire system design can be subdivided into stages as:

- Rule base extraction using GA.
- Rule transmission through UART to FPGA.
- Hardware architecture of FLC.

All the above stages are discussed in the following sections.

Rule base extraction

The most important step in designing of an FLC involves the rule base extraction. The designing can be done either using expert's knowledge or using the available experimental data sheet. The benefits of the use of the data sheet and the constraints of relying on expert's knowledge have been explained before. Rule extraction using data sheet is used for the training of the system.

Different algorithms can be used for initial rule extraction. Fuzzy C-Means (FCM), Hard C-Means (HCM), K-Means algorithms [16], etc., are most widely used algorithms for this purpose. In this paper, we have used K-Means clustering algorithm for rule base designing. The number of clusters equals the number of rules in the rule base. After fixing the number of rules as per requirement, clustering is done to generate rules randomly out of the datasheet at each run.

Genetic algorithm is used to optimize the rule base. The parameters of GA are set as per the required accuracy level of the output. It also depends mostly on the capabilities and constraints of the hardware, doing the computations. Time efficiency, memory efficiency and hardware required for the computation purpose play the most vital role in defining the parameters.

The rule base extraction can be explained under two headings,

1. Initial rule base design.
2. Optimization using genetic algorithm.

Initial rule base design

Initially we need to design initial rule bases equal to the population size of the GA as part of GA parameters. The rule base size is the same for all the designed rule bases. For the designing of each of the initial rule base, K-Means clustering algorithm is being used. To design 'c' number of rules in each of the rule bases, 'c' number of clusters have to be designed. To design one initial rule base in this algorithm, initially 'c' number of data points are chosen randomly, to accommodate flexibility in the designing. They are chosen as the initial cluster centers for each of the clusters. Then the Euclidian distance of each of the data point in the data sheet is calculated from each of the rules, i.e., the cluster centers. The data point is included to that cluster whose center has the minimum distance from the data point. Thus, all the data points are distributed amongst all the clusters.

After clustering, the initial rule base is modified to converge the cluster centers. In each of the clusters, the feature-wise mean is taken for all the data points belonging to that cluster, and these mean values are set as the new cluster centers for the particular cluster. Again clustering is done in the same way as above, taking the modified cluster centers as the new rule base. This process continues till the value wise difference between all the points in the current rule base and the previous rule base comes below a pre-defined threshold, Which states the final convergence of the rule base values. Thus, a final clustered rule base is designed.

Triangular membership function is chosen as the MF for the designed algorithm. Each of the points in the rule base form the center of the corresponding triangular MF. In each of the final clusters, the feature-wise minimum and maximum are selected, which are set as the two end points for the corresponding triangular MF for that particular rule. Let,

C = Number of rules to be designed in a rule base, and $1 \leq \mathrm{i} \leq \mathrm{c}$,
n = Number of data points in the data set, and $1 \leq k \leq n$,
m = Number of features in each data point, and $1 \leq j \leq m$,
D_k = kth data point in the data set = $\lfloor D_{k,j} \rfloor = \lfloor D_{k,1}, D_{k,2} \ldots D_{k,m} \rfloor$
R_i = ith data point in the data set = $\lfloor R_{i,j} \rfloor = \lfloor R_{i,1}, D_{i,2} \ldots D_{i,m} \rfloor$
C_i = ith cluster,
$d_{k,i}$ = distance of D_k from R_i, i.e the cluster center C_i

Then,

$$D_k \in C_i \Leftrightarrow \min_{i=1}^{c} \left\{ \sqrt{\sum_{j=1}^{m} (D_{k,j} - R_{i,j})^2} \right\} = d_{k,i} \tag{1}$$

After clustering, number of data points in C_i, and $1 \leq l \leq p$ then the cluster has the structure like Fig. 2.

A genetic algorithm for tuning of the rule base

Based on the initial designed rule base, the output values are calculated for all the data points in the training data sheet and the sum of all squared errors are calculated. The total error is taken as the objective function for tuning of the rule base. Real-coded GA has been used for the optimization. The GA has been designed to deal with this situation where, the antecedents and the consequent all are membership functions. The proposed algorithm has been used for triangular MFs but it can also be extended to any other type of MF. The parameters of GA are set as per the design requirements.

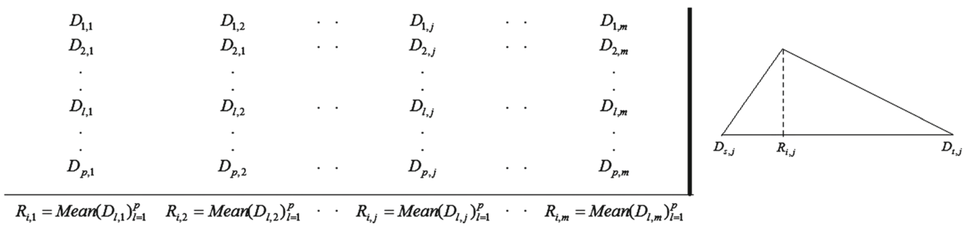

$D_{1,1}$ $D_{1,2}$ $D_{1,j}$ $D_{1,m}$
$D_{2,1}$ $D_{2,1}$ $D_{2,j}$ $D_{2,m}$
$D_{l,1}$ $D_{l,2}$ $D_{l,j}$ $D_{l,m}$
$D_{p,1}$ $D_{p,2}$ $D_{p,j}$ $D_{p,m}$
$R_{i,1} = Mean(D_{l,1})_{l=1}^{p}$ $R_{i,2} = Mean(D_{l,2})_{l=1}^{p}$ $R_{i,j} = Mean(D_{l,j})_{l=1}^{p}$ $R_{i,m} = Mean(D_{l,m})_{l=1}^{p}$
$D_{s,j}$ $R_{i,j}$ $D_{t,j}$

Fig. 2 Rule base design

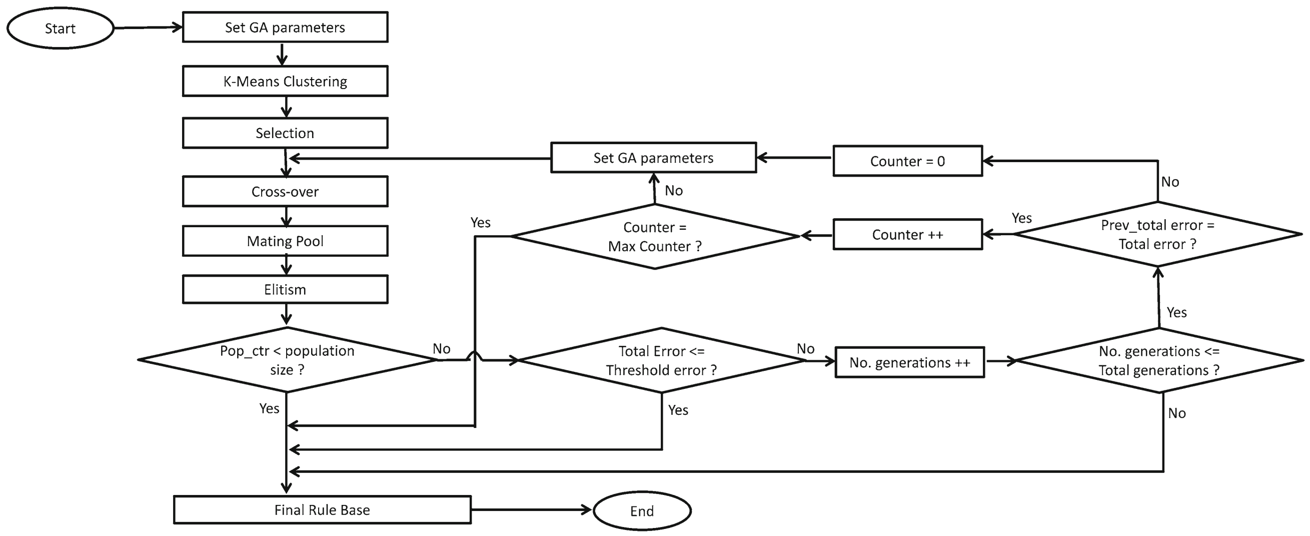


Fig. 3 Flow diagram showing the GA-based optimization of the fuzzy rule base

Figure 3 shows the structure of GA used in this paper. The steps of GA are explained below.

Step 1: selection

As per the algorithm used in this paper, the population size has to be even. The clustering algorithm and initial rule base designing method explained above have to be used to generate initial rule bases for each population. Thus, each population corresponds to one rule base for which total squared error can be calculated for all the data points in the training data sheet. Here, the objective is to reduce the total squared error.

Step 2: crossover

Total squared errors are sorted in ascending order, so the first one corresponds to the best rule base, i.e., rule base with a minimum squared error. The roulette wheel technique is used to select the populations for crossover purpose. The purpose of a crossover is to generate new children from their parents. In this case, both the parents are rule bases and the crossed-over children are also rule bases. A crossover probability is chosen as a design parameter. A number was chosen randomly between 0 and 1, if its value is less than the crossover probability then crossover will be done, else no crossover will take place. The concept of crossover is the better parent gives best children. Total number of crossovers will be population size/2. During each crossover one parent is from the first half of the population. The corresponding parent with whom crossover will happen is chosen randomly from Roulette wheel.

After each crossover two new children are generated. If no crossover is done between the two parents, then parents are transferred to their children. Thus, the total number of children generated after crossover equals the population size. The technique used for crossover is given below.

Crossover is done between the two populations, i.e., between two rule bases. For triangular MFs, only the peaks of the triangles which construct the rule base take part in the crossover process. Suppose the two triangles which take part in crossover have the base points given by [a_1, m_1, b_1] and [a_2, m_2, b_2], then d_1, d_1 are calculated as below.

$$d_1 = \frac{m_1 - a_1}{b_1 - a_1}, \quad d_2 = \frac{m_2 - a_2}{b_2 - a_2} \tag{2}$$

These d_1 and d_2 values were then interchanged between the 2 triangles, i.e., the new value of m_1, m_2 are

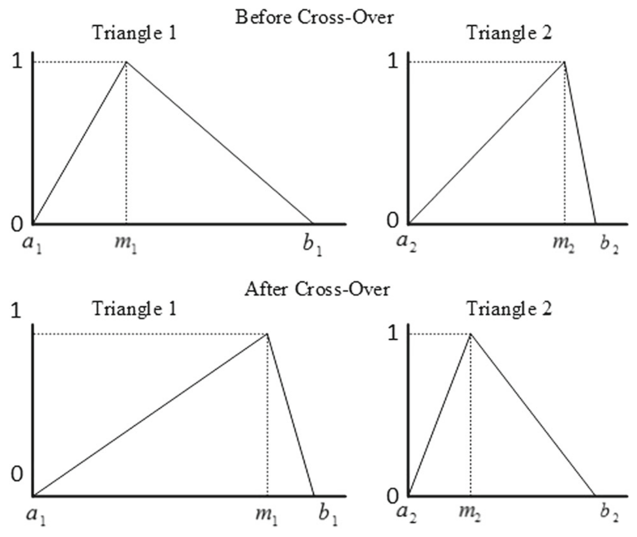


Fig. 4 Shape of membership functions before and after crossover

$$m_1 = a_1 + d_2 * (b_1 - a_1), \quad m_2 = a_2 + d_1 * (b_2 - a_2) \quad (3)$$

The end points of the triangle remain unchanged. The shape of membership functions before and after the crossover is shown in Fig. 4.

Step 3: mutation

The newly generated triangles are again modified slightly expecting for a better rule base. A mutation probability and a mutation fraction are set by the designer as a design parameter. Maximum number of points that can be mutated is equal to mutation probability times the total number of points. In our case, the triangle centers and the boundary points may get mutated to give a better rule base. The above algorithm includes that if the center point of a triangle gets mutated then the boundary points of the triangle will also get mutated to give a new triangle.

A number is chosen randomly between 0 and 1. If that number is below the mutation probability then mutation will take place, otherwise no mutation will occur. The mutation of a triangle described by the points $[a, m, b]$ is done as given below. The minimum distance is calculated, i.e., $d = min(m - a, b - m)$. Then a new value of 'm' is assumed in the neighborhood of 'm' within a given region,

$$([m - d] * \text{mutation}_{\text{f}}\text{actor}, [m + d] * \text{mutation}_{\text{f}}\text{actor}) \quad (4)$$

Similarly the boundary points of the triangle also get mutated. 'a' was assumed within a region given by,

$$(a - [m - a] * mutation_f actor,$$
$$b + [m - b] * mutation_f actor) \quad (5)$$

and 'b' is assumed within a region given by the value of 'm' used in the above boundary conditions which is actually the old value of 'm' before getting mutated.

Step 4: elitism

The total squared errors for all the populations, i.e., for all the rule bases generated before crossover, after crossover and after mutation are calculated. The rule bases for next generation of GA are chosen from the mating pool, based upon the minimum total squared error, i.e., all the rule bases are sorted in ascending manner of their total squared error, and the number of rule bases carried forward to the next generation equals the population size of the GA. These new populations are again modified through the above procedure in subsequent generations to give an optimum rule base with a minimum total squared error. The plot of optimized rule base is drawn in Fig. 5 for input X1, input X2 and output Y. This generation process terminates if,

- The number of generations set by the designer are over.
- The total squared error comes down below the required error level.
- The total squared error does not get modified over a number of generations, i.e., no improvement is seen in the rule base over a number of generations.

Rule base transmission

After the rule base optimization, the system parameters are sent through serial communication over the UART to the FPGA and vice versa using the Matlab GUI as given in Fig. 6. The transfer of all optimized rules, input values and control information follows fixed point representation, the following points explain the data representation and total number of bytes to finish the communication process.

1. Data to be sent are multiplied by 256, i.e., 8-bit shift in binary value.
2. The obtained result is rounded to nearest integer value.
3. This value is represented in 16 bits Q8.8 format and sent over the UART in chunks of bytes with higher byte followed by lower byte.
4. Initially control registers are transmitted, which indicates the availability of the crisp input values and the number of rules and also initiates the hardware FLC process.
5. The transmission of data points of reduced rules begins from the reverse direction, i.e., the first upper value of the triangle of the rule was sent. Then the center value of the rule followed by the lower value.

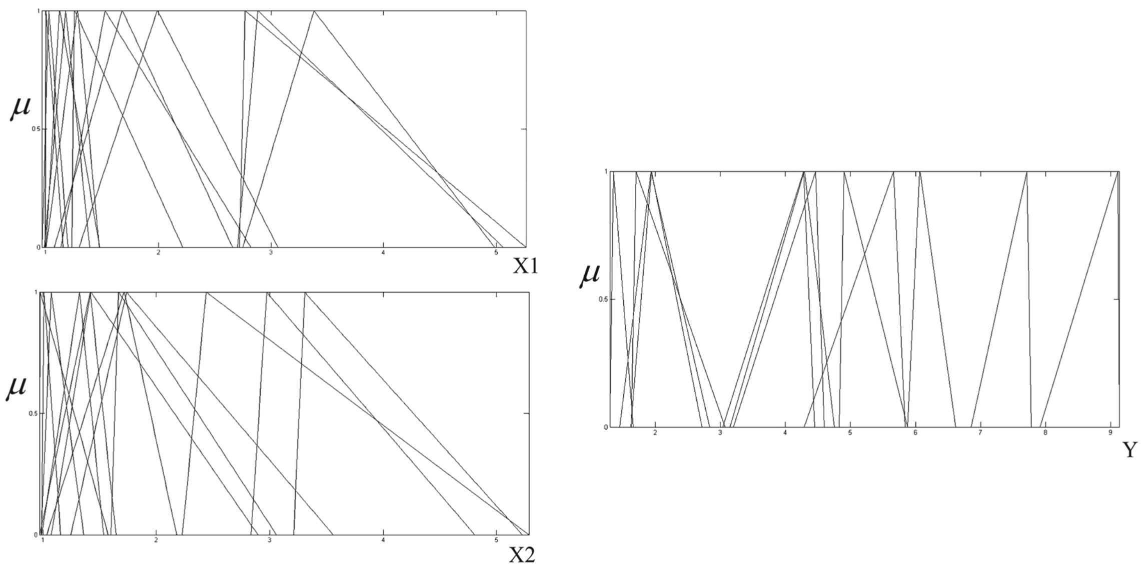
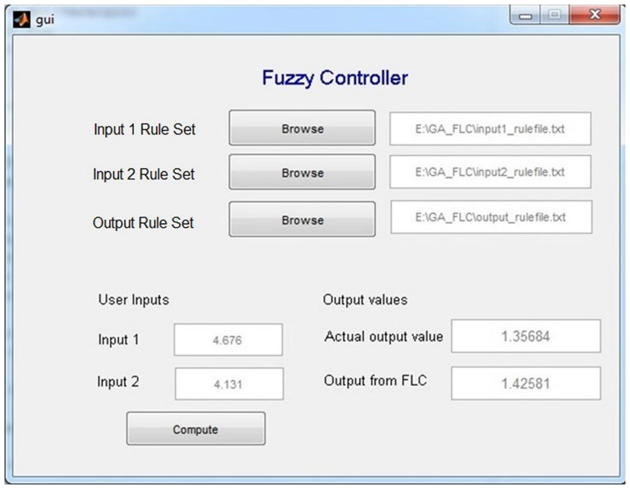


Fig. 5 Reduced rule base with GA optimization



Fuzzy Controller

Input 1 Rule Set E:\GA_FLC\input1_rulefile.txt
Input 2 Rule Set E:\GA_FLC\input2_rulefile.txt
Output Rule Set E:\GA_FLC\output_rulefile.txt

User Inputs		Output values	
Input 1	4.676	Actual output value	1.35684
Input 2	4.131	Output from FLC	1.42581

Fig. 6 Designed GUI using MATLAB for rule transmission to FPGA and computed value reception from FPGA

6. Once the first rule is sent over the UART, the second rule was sent followed by the third rule. This sequence was followed till the final rule was sent.
7. After the rule base was sent successfully, high byte of nth crisp input was sent, then its low byte. All the crisp input values were sent as per the number of input parameters.
8. Total number of points = No. of rules * No. of points required to represent each membership function * (No. of inputs + No. of outputs).
9. Total number of bytes to be transferred a = Total number of points 2.
10. For examples, let us design a system for 2 inputs and 1 output. Triangular membership functions are used to design 8 rules in the rule base both for input and output side. 3 points are required to represent each triangle.
11. Total number of data points to be transferred = 8 * 3 * (2 + 1) = 72.
12. Total number of bytes(In Q8.8 format) to be transferred = 72 * 2 = 144.
13. After 144 bytes have been captured, the FLC treats the next pair of bytes as inputs. That finishes the building process.

Hardware architecture of the FLC

The design goal was to enable extraction of synthesized rule base from a more capable hardware platform onto a dedicated implementation platform for the fuzzy system. The interface was chosen as UART due to low speed and simplicity which shall reduce debugging time. The choice of components for the architecture was dictated by the terms imposed by the HDL as well as the synthesis software. A word size of 16 bits was considered as the numbers were represented in Q8.8 format. Rule base representation was addressed following the triangular nature of membership functions. As triangular functions are used, a set of three points is required to represent each function. The software transmits these points as a set of two bytes over UART. The membership function points of the rules are stored in RAM locations after the UART on FPGA has captured the bytes. Following the rule base extraction, the crisp input values are sent to hardware and are also stored in RAM locations. Once new crisp inputs are transferred, the software writes value 0xA3H to RAM location 0x0H, which is the control word as mentioned in Table 1. The FLC keeps polling this value, upon reading 0xA3 the FSM initiates the

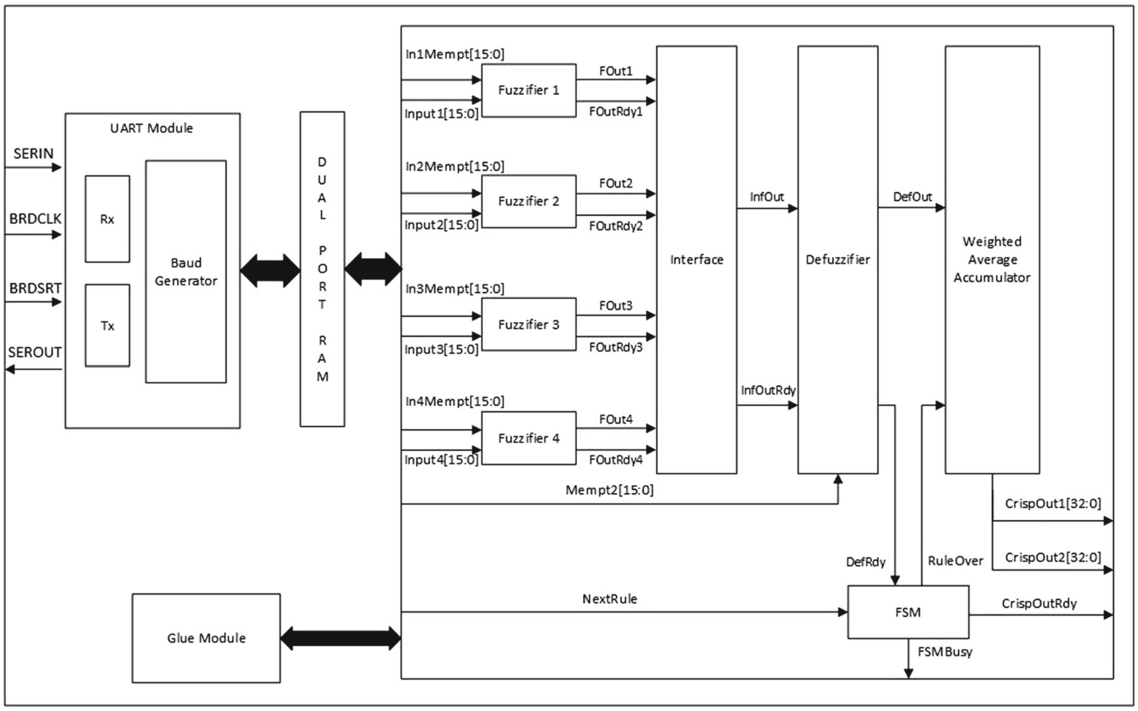


Fig. 7 Block diagram of fuzzy inference module

FLC process. The process ends after computing crisp output value software acknowledges this completion by reading 0xA1 data on RAM location 0x0H, and initiates the fresh FLC process.

A dual-port RAM is used by its ports A and B interfaced to the UART and the FLC modules, respectively. The Xilinx core generator tool was used to generate the core of the block RAM module. The functionality of dual ports purges bus sharing issues. The address on port A is set to 0x00 and increments twice for each membership point or crisp input (2 bytes). Arrival of data over UART triggers address increment and memory write operation. Figure 7 shows the block diagram of the fuzzy inference module.

The rule base is ready for inference processing. The address is set to 0x00H and incremented till it reaches NoOfBytes. Evidently control registers are to be read first. Then the membership function points are read rule-wise. The software transmits the data in a sequence such that the last rule comes in last transaction. The location NoOfBytes + 1 to NoOfBytes + 4 are used to store crisp output if outputs are 1 and NoOfBytes + 1 to NoOfBytes + 8 are used to store crisp outputs if outputs are 2.

$$No\ of\ bytes = 2 + 2 * No\ of\ inputs + 2 * 3 * No\ of\ rules * (No\ of\ inputs + No\ of\ outputs) \quad (6)$$

The weighted average method is chosen for defuzzification for its simple hardware structure. Here FLC is designed to evaluate the outputs rule-wise and accumulate them. It reads the points for the one rule only at a time and computes the output before reading the next set of points, finally all defuzzified values are accumulated using an accumulator, and this proposed architecture reduces the logic utilization of final FLC. Finite state machine used to control each process in FLC is shown in Fig. 8 and its state transition is shown in Table 2.

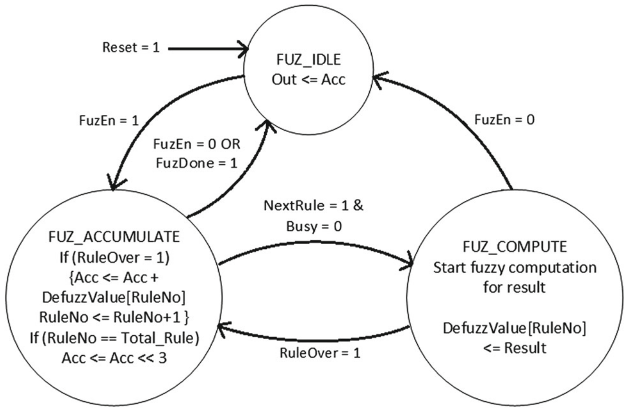


Fig. 8 FLC finite state machine

Mamdani inference method is used here to implement inference processing. The FIP module consists of three submodules and control logic. There is a 2-byte register file consisting of 8 registers for storing each rule's output. A 5-byte accumulator is present for multiplication and accumulation. The sub-modules are,

Table 2 State transition

Present state	Next state	Transition signal	Description
Fuz_IDLE	Fuz_ACCUMULATE	FuzzyEn	Initialize accumulation process
Fuz_ACCUMULATE	Fuz_COMPUTE	NextRule & !Busy	The system waits for a new set of rules and input points to trigger FIP
Fuz_ACCUMULATE	Fuz_IDLE	!FuzEn | FuzDone	Once FIP is finished it goes back IDLE state for new FIP
Fuz_COMPUTE	Fuz_ACCUMULATE	RuleOver	Generates enable signals to start FIP and updates the accumulator register with new rule output
Fuz_COMPUTE	Fuz_IDLE	!FuzEn	In-between process if software interrupts to stop process state goes back to IDLE

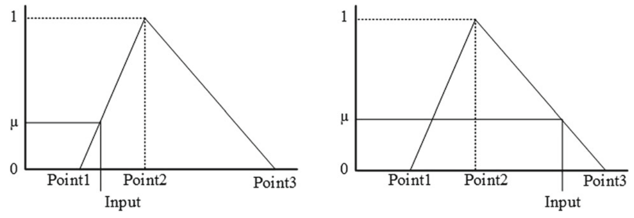


Fig. 9 Calculation of membership values

1. Fuzzification unit for making crisp quantity fuzzy.
2. An inference engine unit to compute overall control output based on individual contribution of each rule in the reduced rule base.
3. The defuzzification unit which converts fuzzification quantity to the precise quantity.

Figure 9 shows the calculation of fuzzified values in hardware by evaluating the crisp input membership degree using membership functions by:
if $Input \geq Point_1$ and $Input \leq Point_2$

$$\mu = \frac{Input - Point_2}{Point_2 - Point_1} \tag{7}$$

If $Input \geq Point_2$ and $Input \leq Point_3$

$$\mu = \frac{Input - Point_2}{Point_2 - Point_1} \tag{8}$$

The max–min composition of inference model proposed by Mamdani [17] illustrated in Fig. 10 and Table 3 is applied for hardware realization of inference module. The inference module outputs are defined by:

$$\mu O_{m1}^{r1}(y) = min[\mu A1_{j_1}(X1), \mu A2_{k_1}(X2)] \tag{9}$$

$$\mu O_{m1}^{r3}(y) = min[\mu A1_{j_3}(X1), \mu A2_{k_3}(X2)] \tag{10}$$

$$\mu O_{m1}^{r1\&r3}(y) = max\{min[\mu A1_{j_1}(X1), \mu A2_{k_1}(X2)], min[\mu A1_{j_3}(X1), \mu A2_{k_3}(X2)]\} \tag{11}$$

$$\mu O_{m2}^{r2}(y) = min[\mu A2_{j_2}(X1), \mu A2_{k_2}(X2)] \tag{12}$$

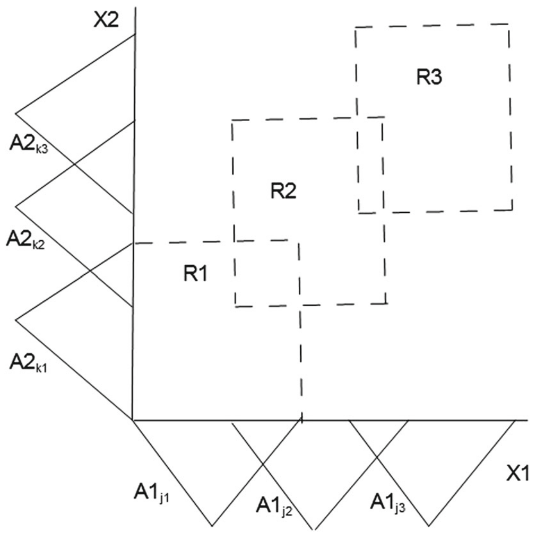


Fig. 10 Process of fuzzy controller

Table 3 Rule base of simple FLC

X1	X2	Rule	Y
$A1_{j_1}$	$A2_{k_1}$	r1	O_{m1}
$A1_{j_2}$	$A2_{k_2}$	r2	O_{m2}
$A1_{j_3}$	$A2_{k_3}$	r3	O_{m1}

Instead of composite output membership function we used weighted average defuzzification (Fig. 11) for its simplicity in hardware implementation as it needs only clipped or scaled output membership functions. This method simply takes peak value of each clipped/scaled output fuzzy sets and builds weighted sum of these peak values given by:

$$Y* = \frac{\mu O_{m1} * P1 + \mu O_{m2}}{P1 + P2} \tag{13}$$

The implemented block diagrams of fuzzifier, inference and defuzzifier and their top modules in an FPGA are shown in Fig. 12.

Experiment and implementation results

Hang data function [2 input 1 output system]

Hang data function is a test case mathematical function with two inputs and one output. The mathematical representation of the function is given by

$$y = \{1 + x_1^{-1.5} + x_2^{-2.5}\}^2 \quad (14)$$

where

$1 \leq x_1 \leq 5,$

and

$1 \leq x_2 \leq 5$

We obtained 800 input–output data by sampling the input range $x_1, x_2 \in \{1, 5\}$. Rule base had been generated for this test case and it was optimized using GA. In this paper, fuzzy rule base has been designed for hang function by generating a training datasheet using the basic equation. The designed dataset is used for training and tuning of the rule base while using GA as the tuning algorithm. A list of GA and fuzzy parameters are provided as shown in Table 4, which are to be taken into consideration in the designing of the fuzzy optimized system for Hang data function. Results before and after optimizations of the rule base are as follows:

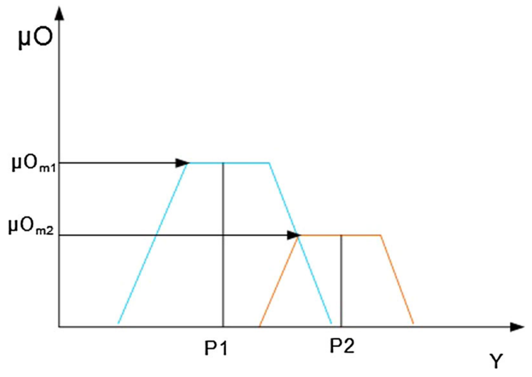


Fig. 11 Defuzzification

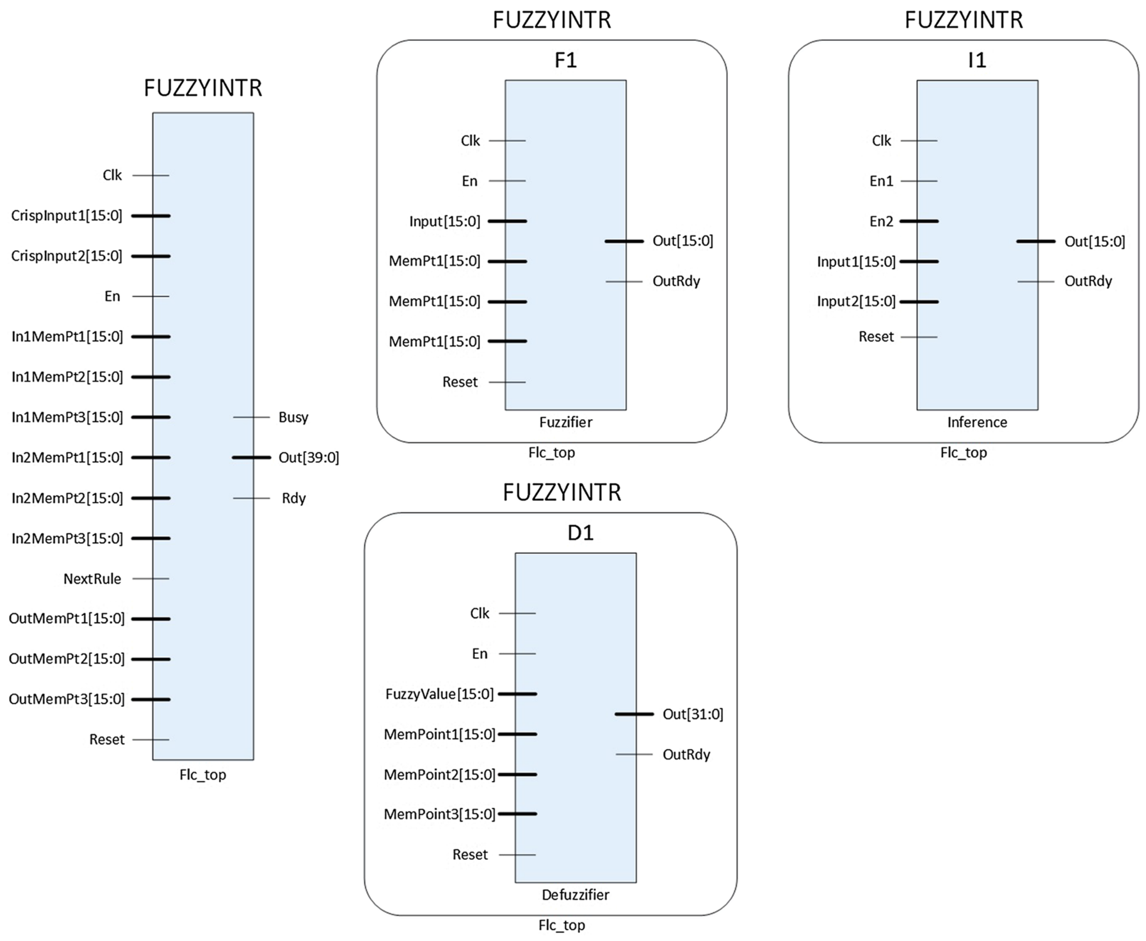


Fig. 12 Block diagrams of fuzzy inference processing top module and fuzzifier, inference and defuzzifier

Table 4 GA-FLC system manual to generate optimized rules for hang data function

S. No.	GA-FLC system parameters	Value
1	Number of points in the data sheet	800
2	Number of features	3
3	Number of designed rules	12
4	Number of populations used for GA optimization	6
5	GA crossover percentage	0.8
6	GA mutation percentage	0.3
7	Number of GA generations	1000
8	GA error threshold value	0.001
9	Samples for integration in defuzzification	1000

1. Total mean square error for initial rule base = 283
2. Total mean square error for final optimized rule base = 23

Actual surface plot for hang data function is in Fig. 13. Figure 14 show the surface plot of hang data function after the rule base optimized from the GA where the centroid method is used for defuzzification. The FPGA implemented GA-FLC surface plot is shown in Fig. 15 where the weighted average method is used to reduce the hardware resource consumption for defuzzification. The comparison of GA-FLC with 8 rules and artificial Neuro fuzzy inference system (ANFIS) with 49 rules and 10 epochs is shown in Fig. 16, where we can observe that error generated using the GA-FLC gives better results with its optimization advantages.

Chaotic time series [4 input 1 output system]

Chaotic processes are the type of processes which have got a disordered mechanism of functioning. These processes have acquired this behavior because of the presence of some kind of positive feedback. The analysis of this sort of processes against time results in a form of a random time series which we call chaotic time series. These systems are never com-

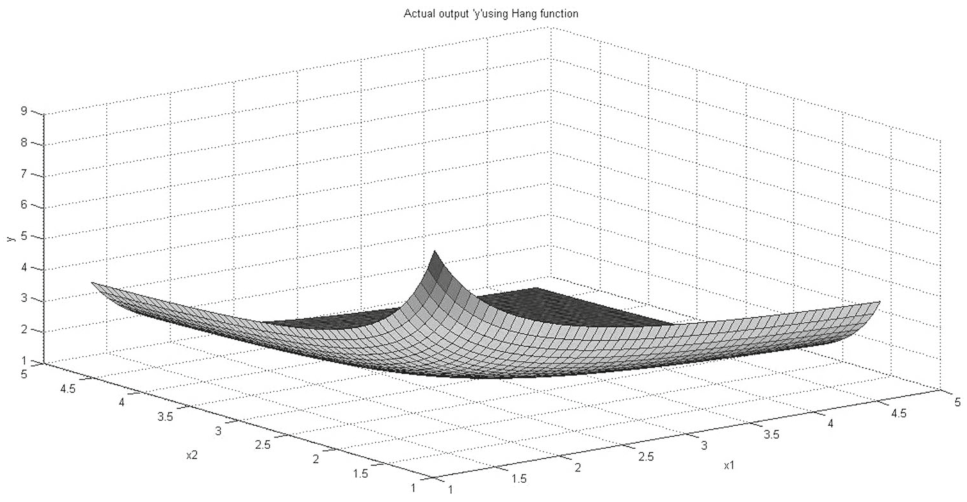


Fig. 13 Actual surface plot

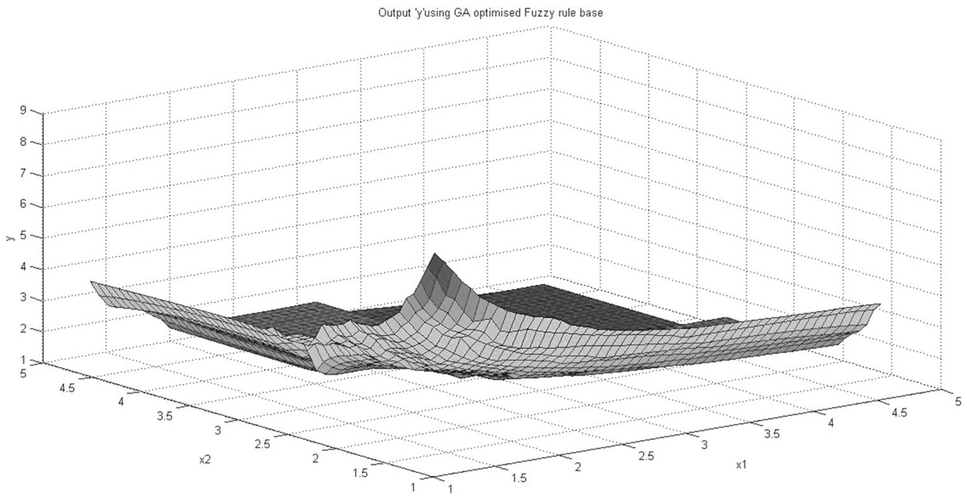


Fig. 14 Using rule base obtained from GA

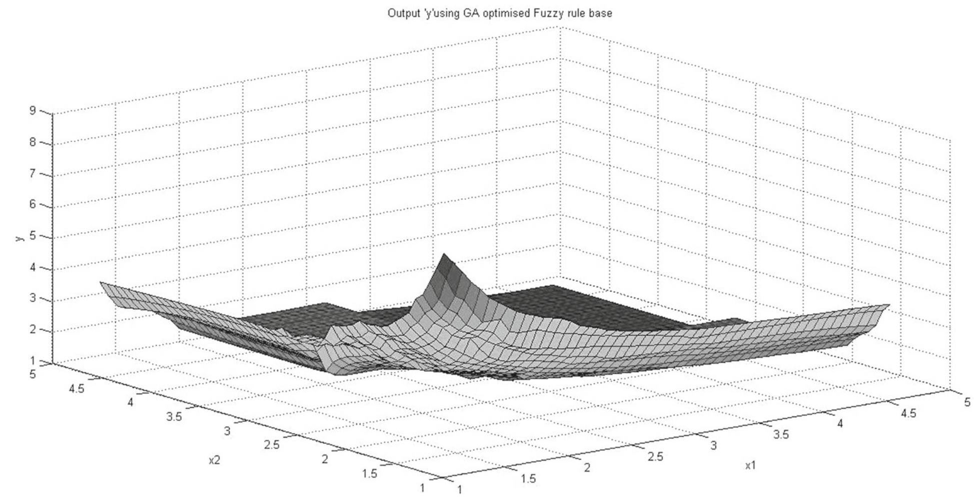


Fig. 15 Using GA-FLC on FPGA

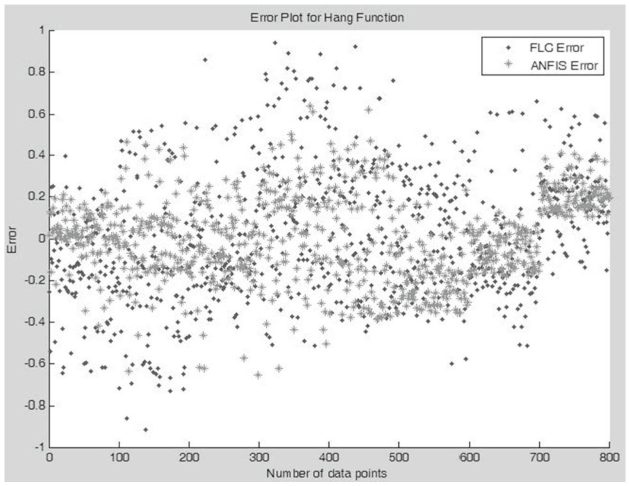


Fig. 16 GA-FLC versus ANFIS error plot

pletely predictable because of feedback the simulation and the real series will always rapidly diverge.

Let $x(t)$ be a chaotic time series. If t_0 = start time, Δt = time interval, then at time point $t_k = t_0 + \Delta t * k$, $(0 \leq k \leq n)$. The chaotic time series data can be represented as,

$$X = (x_0, x_1, \ldots x_n) = (x(t_0), x(t_1), \ldots x(t_n)) \tag{15}$$

Because of the unpredictability of the chaotic time series, it is not possible to mathematically model the series in terms of equations. Hence, the objective is to design a data predictive model T, to predict the value x'_m of the series at time instance t_m based on the available data set, $\{x_k \mid k \leq m\}$ such that $\mid x_m - x'_m \mid$ is as minimum as possible.

GA-optimized fuzzy rule base algorithm was used to design the model for chaotic time series prediction. The chaotic series taken into consideration had $\Delta t = 6$ and the prediction was done with four previous available data set. The mathematical model is:

Table 5 GA-FLC system manual to generate optimized rules for hang data function

S. No.	GA-FLC system parameters	Value
1	Number of points in the data sheet	1000
2	Number of attributes chosen for prediction	4
3	Number of designed rules	10
4	Number of populations used for GA optimization	10
5	GA crossover percentage	0.8
6	GA mutation percentage	0.3
7	Number of GA generations	1000
8	GA error threshold value	0.001
9	Samples for integration in defuzzification	1000

$$\begin{aligned} x(t+\Delta t) &= F[x(t), x(t-\Delta t), x(t-2\Delta t), x(t-3\Delta t)] \\ \Rightarrow x(t+6) &= F[x(t), x(t-6), x(t-12), x(t-18)] \end{aligned} \tag{16}$$

Parameters chosen for designing of the 4 input–1 output GA-optimized rule base are given in Table 5. After the rule base was designed it was tested with real-time data points and the mean square error obtained for 1000 data points is $5.244 * 10^{-4}$. The optimized fuzzy rule base was sent to the designed FPGA model to predict the data for the same 1000 data points. Comparison of plots are given in Figs. 17 and 18 and the error plot is shown in Fig. 19.

Simulation and hardware implementation

The implementation of the FLC is aboveboard by coding each module in Verilog hardware description language integrated with Xilinx foundation ISE 14.2 tool, which supports ISim (Integrated within ISE) used here for func-

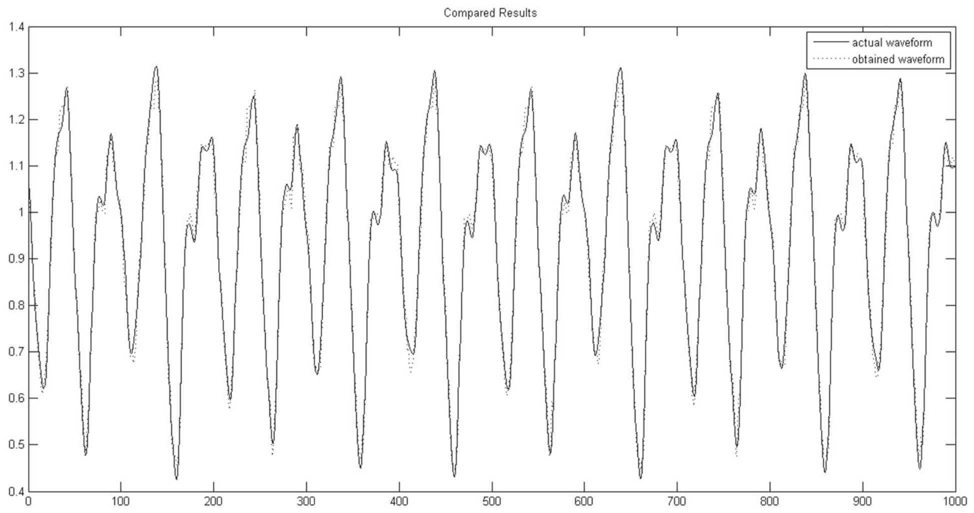


Fig. 17 Comparative plot between desired time series data and GA rule base predicted data

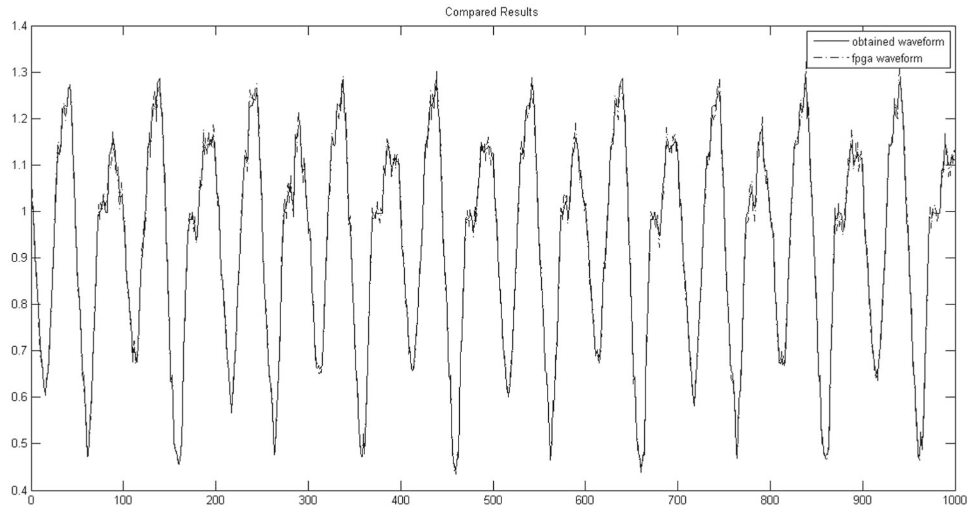


Fig. 18 Comparative plots between GA rule base predicted data and GA-FLC on FPGA

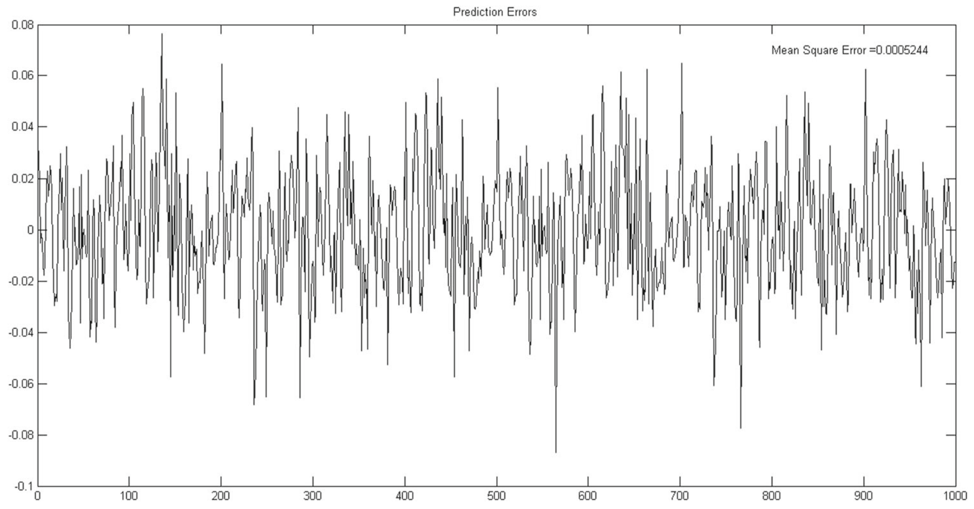


Fig. 19 Prediction errors between desired time series data and GA-predicted data

tional verification. The architecture of the FLC is highly pliable as the parameters of the fuzzy logic controller can be changed by changing registry values and Verilog parameters.

Simulation parameters

1. The UART interface includes receiving and transmits modules that share a single baud generator module. The

baud rate is set by two constants at FLC top modules which are calculated as follows:

$$\text{X\_BAUD\_FREQ} = \frac{(16 * \text{BaudRate})}{(\text{GCD}(\text{GlobalClkFreq}, 16 * \text{BaudRate}))} \quad (17)$$

$$\text{X\_BAUD\_LIMIT} = \frac{\text{GlobalClkFreq}}{(\text{GCD}(\text{GlobalClkFreq}, 16 * \text{BaudRate}))} - \text{X\_BAUD\_FREQ} \quad (18)$$

2. For 100 MHz of board clock in XUPV5 FPGA board, the calculated parameters to set in Verilog is as follows:

   ’define X_BAUD_FREQ 12’H90
   ’define X_BAUD_LIMIT 12’H0ba5

3. The test bench parameterized for total no. of inputs and no. of membership functions and data bus width for each input and membership value is as below:

   ’define NO_OF_INPUTS 3’h2
   ’define NO_OF_MFS 3’h7
   ’define DATA_BUS_WIDTH 6’h10
   ’define DEFUZZY_METHOD 2’h0

Hardware implementation

The functional simulations obtained by ISIM 14.2 are presented in Fig. 20, where the READY signal enables UART transmitter to transmit crisp out data back to MATLAB GUI for display. Figure 21 captured the crisp data transfer from FPGA to MATLAB GUI using Chipscope-Pro debugging tool. The implemented platform chosen in this paper was the Virtex V (LX110T) Xilinx FPGA family included in XUPV5 development board. This FPGA is sufficient enough for implementing all the modules of the FLC addressed in this paper. This was possible as the FPGA contains 17,280 slices and 68 DSP48E slices as well as 296 18kb block RAMs. Table 6 ushered the FPGA utilization to develop the FLC addressed here. The total clocks needed to complete on fuzzy logic inference process (FLIP) with GA reduced 8 rules are

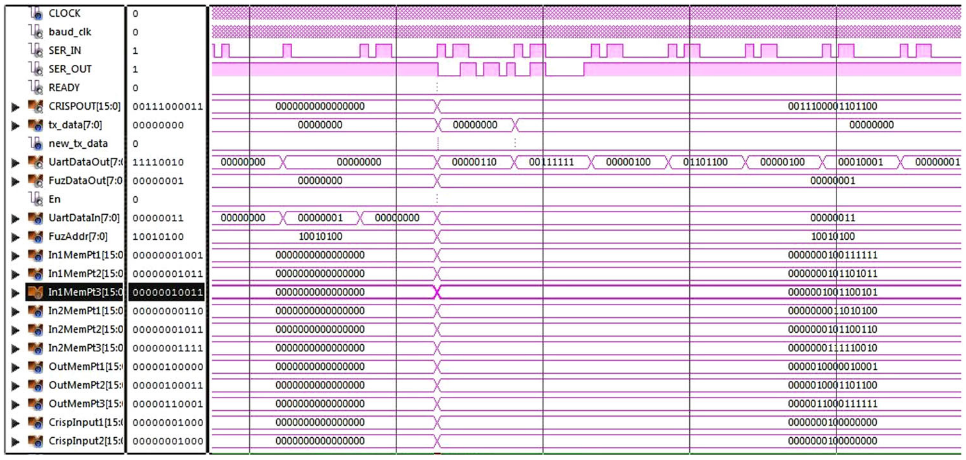


Fig. 20 The functional simulation waveform obtained by ISim 14.2

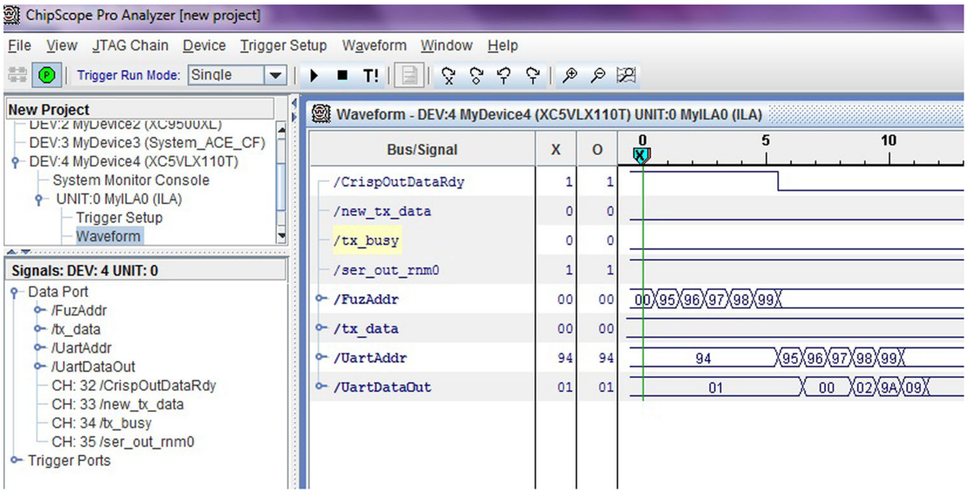


Fig. 21 Crisp data output from FPGA using UART captured on Chipscope-Pro tool

Table 6 Device utilization summary

Experiment	4 input–1 output system	2 input–1 output system
Selected device	xc5vlx110t-2-ff1136	xc5vlx110t-2-ff1136
Maximum frequency	213.413 MHz	246.819 MHz
Number of slices	1456 out of 69,120 2 %	1149 out of 69,120 1 %
Number of 4 input LUTs	1919 out of 69,120 2%	1294 out of 69,120 1 %
Number of bonded IOBs	22 of 640 3 %	22 of 640 3 %
Number of MULT18X18s	1 out of 640 1 %	1 out of 640 1 %
Number of GCLKs	1 out of 32 6 %	1 out of 32 6 %

325 clocks with 247 MHz clock frequency, this means that the total time required to complete one FLIP to generate output is 1.315 μs, it is equal to 760,000 (760KFLIPS) fuzzy logic inference outputs which can be made using the current design.

The GA-FLC using FPGA is proved to be a good framework, which can be easily used to develop fuzzy logic applications within a short period and the main functional units can be reused without modification and user can only concentrate to provide input–output data sheet, membership values and the number of rules, which increases the design efficiency.

Conclusion

A novel approach towards the rule base synthesis for fuzzy systems was taken using an evolutionary algorithm like genetic algorithm, for convergence of the rule base to provide better results. Comparison with an ANFIS model in MATLAB which used 49 rules and ran for 100 iterations shows that this novel method provides better results with lesser rules. Further, there is no need for an expert to design the system. An accurate data set of the system/plant under discussion is required only for generating the rule base. The following are the benefits of the algorithm proposed in this paper:

(i) The user can design required number of rules as per the system memory and time constraints for better response of the system.
(ii) The proposed GA rule base optimization system is easily configured.
(iii) Actual process changes can be easily incorporated by redesigning the rule base in very less time.
(iv) Rule bases can be designed as per the system tolerance limit.
(v) As we have mentioned the problems with existing fuzzy systems, our process handles the above issues with automatic rule base designing and auto-optimization process.
(vi) The fuzzy rule base is scalable, i.e., any change in the actual process behavior can be taken into consideration easily without the intervention of any person and extra brainstorming. Number of rules and iterations can be increased to incorporate system tolerance bandwidth.

The hardware extracts the rule base once it is synthesized by the software. This is done over UART. Reprogrammability of the rule base makes the system more flexible. The hardware design supports tuning of membership functions and rule base tables for different applications. The future efforts will be directed to accommodate the use of other membership functions (Gaussian, sigmoid, etc.), implication methods, and defuzzification methods.

Acknowledgments The authors would like to thank the Department of ECE, NIT Rourkela, India for providing the material and boards to perform the present work.

References

1. Khrais S, Al-Hawari T, Al-Araidah O (2011) A fuzzy logic application for selecting layered manufacturing techniques. Expert Syst Appl 38(8):10286–10291
2. Malik H, Mahto T, Kr BA, Kr M, Jarial RK (2012) Fuzzy-logic applications in transformer diagnosis using individual and total dissolved key gas concentrations. J Electr Eng 12(3):202–206
3. Nair MS, Lakshmanan R, Wilscy M, Tatavarti R (2011) Satellite image processing for oceanic applications using fuzzy logic. Int J Intell Syst Technol Appl 10(3):289–301
4. Ropero J, Leon C, Carrasco A, Gomez A, Rivera O (2011) Fuzzy logic applications for knowledge discovery: a survey. Int J Adv Comput Technol 3(6):187–198
5. Guillaume S (2001) Designing fuzzy inference systems from data: an interpretability-oriented review. IEEE Trans Fuzzy Syst 9(3):426–443
6. Orriols-Puig A, Martinez-Lpez FJ, Casillas J, Lee N (2013) A soft-computing-based method for the automatic discovery of fuzzy rules in databases: uses for academic research and management support in marketing. J Bus Res 66(9):1332–1337

7. Dounis AI, Lefas CC, Argiriou A (1995) Knowledge-based versus classical control for solar-building designs. Appl Energy 50(4):281–292
8. Nomura H (1992) Self tuning method fuzzy reasoning by genetic algorithm. In: Proc. of the Int'l Fuzzy systems and intellignet control, pp 251–256
9. Glorennec PY (1993) Roubens M, Lowen R (eds) Adaptive fuzzy control, 12th edn. Springer, Netherlands
10. Siarry P, Guely F (1993) Gradient descent method for optimizing various fuzzy rule bases. In: 2nd IEEE Int. Conf. Fuzzy Syst, San Francisco, p 124146
11. Cordon O, Herrera F (1999) A two-stage evolutionary process for designing TS fuzzy rule-based systems. IEEE Trans Syst Man Cybern Part B Cybern 29(6):703–715
12. Wang H, Kwong S, Jin Y, Wei W, Man K-F (2005) Agent-based evolutionary approach for interpretable. IEEE Trans Systems Man Cybern 35(2):143–155
13. Fu Y, Li H, Kaye ME (2010) Hardware/software codesign for a fuzzy autonomous road-following system. IEEE Trans Syst Man Cybern Part C Appl Rev 40(6):690–696
14. Jammu BR, Patra SK, Mahapatra KK (2013) VLSI architecture of reduced rule base inference for run-time configurable fuzzy logic controllers. In: 7th International conference on bio-inspired computing: theories and applications, BIC-TA 2012. Springer Verlag, Heidelberg, pp 77–88
15. Cabrera AJ, Baturone I, Moreno-Velo FJ, Brox M, Sanchez-Solano S (2007) FPGA implementation of embedded fuzzy controllers. IEEE Trans Ind Electron 5(2):55–67
16. Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv 31(3):264–323
17. Mamdani EH, Assilian S (1975) An experiment in linguistic synthesis with a fuzzy logic contoller. Int J Man Mach Stud 7(1):1–13