Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/taut20

Development of embedded fuzzy control using reconfigurable FPGA technology

Ayman A. Nada & Mona A. Bayoumi

To cite this article: Ayman A. Nada & Mona A. Bayoumi (2024) Development of embedded fuzzy control using reconfigurable FPGA technology, Automatika, 65:2, 609-626, DOI: 10.1080/00051144.2024.2313904

To link to this article: https://doi.org/10.1080/00051144.2024.2313904

Published online: 14 Feb 2024.

AUTOMATIKA
2024, VOL. 65, NO. 2, 609–626
https://doi.org/10.1080/00051144.2024.2313904

RESEARCH ARTICLE

OPEN ACCESS Check for updates

Development of embedded fuzzy control using reconfigurable FPGA technology

Ayman A. Nada [a] and Mona A. Bayoumi[b]

[a]Mechatronics and Robotics Department, School of Innovative Design Engineering, Egypt-Japan University of Science and Technology E-JUST, Alexandria, Egypt; [b]Electrical Engineering Department, Benha Faculty of Engineering, Benha University, Benha, Egypt

ABSTRACT

The purpose of this work is to investigate the construction of fuzzy embedded control systems combining fast execution and parallel processing capabilities provided by Field Programmable Gate Arrays (FPGAs) and Reconfigurable Inputs/Outputs (RIO) chips. A fixed-point fuzzy controller is developed and implemented on a fast mechatronic system with high-speed control and high channel count on an FPGA target. This paper provides a brief introduction to deploying Fuzzy Logic Control (FLC) methods using RIO-FPGA technology. It suggests a technique for implementing the three stages that constitute the FLC and the PD-like FLC and PID-like FLC structures into practice. Controllers with 1 and 2 degrees of freedom are developed and tested experimentally. Parallel loops, key challenging advantage of LabVIEW programming, are utilized for decoding feedback signals, generating pulse trains for actuator's drivers and for calculation of control gains. An NI-SbRIO board that combines deployable devices with a real-time processor, a re-configurable FPGA, and analogue and digital input–output ports is used. The experimental work demonstrates the significant enhancement of implementing reconfigurable embedded fuzzy control upon such mechatronic systems.

ARTICLE HISTORY
Received 11 September 2022
Accepted 25 January 2024

KEYWORDS
Embedded systems; FPGA; fuzzy logic; mechatronic systems

1. Introduction

Classical and model-based control methods need a mathematical description of the system; however, fuzzy controller considers systems as a black box that is a noteworthy prominence of the fuzzy controller in comparison with other traditional methods [1]. Unlike with conventional, computationally expensive control algorithms, Fuzzy Logic control does not need a fast processor or very precise measuring instruments to get admissible results. At lower costs, Fuzzy Logic Control (FLC) provides a simple way to arrive at a definite output based upon inexact, uncertain, imprecise, noisy, or missing input information [2, 3]. This is a distinguishing feature of Artificial Intelligence (AI) that makes quick and urgent judgment actions, without the need to be absolutely certain of the input.

The most evident feature of the fuzzy controller is the experimental use of expert's knowledge. Along with this advantage, it has a big problem. These rules and amounts might not be optimized or be completely wrong. That is why the thought was to utilize implementation mechanisms that could be reformulated upon application, i.e. re-configurable controller. Regardless of this disadvantage, it also required to provide a technique for optimal implementation of FLC upon embedded and fast mechatronic systems. In such applications, it is required to overcome the complexities of the real-time implementation of the controller, especially for nonlinear systems.

Multiple stages are required to execute fuzzy computation. The total computational time is based upon the complexity of the given system. For a small, simple control system, the computation time is not a significant factor. In the majority of situations, the computation is complex, indicating the presence of several controls within the entire system. Moreover, parallelism is frequently inherent in these computations [4].

A review of the relevant literature reveals that real-time control necessitates the parallel construction of algorithms that can be achieved through splitting of instructions that are deemed independent of each other among multiple processing elements [5–8]. Various utilization techniques have been identified in the literature. First, the utilization of multi-threads (OpenMP or OpenMPI) for parallel implementation in multiprocessors may lead to an energy overhead due to the utilization of multiple large processing elements. Chantrapornchai et al. [4] demonstrated that OpenMP has certain overhead and the dependency between iterations may prohibit the iteration parallelization. Second, the parallelized processors encompass a range of processing units, such as Very-Large-Instruction-Word processors, Digital Signal Processing processors

CONTACT Ayman A. Nada ayman.nada@ejust.edu.eg Mechatronics and Robotics Department, School of Innovative Design Engineering, Egypt-Japan University of Science and Technology E-JUST, New Borg El-Arab City, Alexandria 21934, Egypt

and Graphics Processing Units. All of the aforementioned alternatives are architectures that follow the Single-Instruction–Multiple-Data paradigm, which is intended to execute identical operations on numerous data elements concurrently. These devices are specifically optimized for the purpose of executing digital filters. Typically, the cost of each processing element in any of the aforementioned architectures is lower than that of a single processing element in a conventional multiprocessor. However, these architectures tend to have limited compiler support. This form of parallel processing is well-suited for tasks such as image processing and feature recognition. Third, microcontrollers possess the ability to execute compiled C code and, similar to larger processors, they possess the advantage of having built-in memory. Accessing memory will not be as quick because of the serial nature of access, but several microcontrollers working in parallel could speed up the process.

In this context, Field Programmable Gate Array (FPGA) can be considered as a valuable tool for control implementation upon embedded systems due to low consumption of energy, high speed operations and considerable capacity of data storage and can be deliberated for the optimal solution of implementing the intelligent control methods upon such systems [9, 10]. It is proposed that the FPGA and Real-Time modules of LabVIEW software may introduce an efficient hardware for control implementation [11–13]. The powerful FPGA's robust capabilities enables the fast execution speed, parallelism the data processing and the re-configurability that allow for the adaptation of the FLC-FPGA controller design and implementation.

It should be mentioned that whereas microcontrollers are still used in low- to medium speed applications for embedded systems, i.e. require relatively large computation time, FPGAs have made it possible to target high speed applications and use minimal computation time for the controller [14].

On the practical side, LabVIEW setups can be executed in one of two modes: first, as hardware-in-the-loop (HIL), where the host computer controls the real-time mechatronic system's hardware via wireless network protocols that are used to transmit control actions [15–18]. In the second mode, the system can be utilized as a standalone real-time system that executes the control law independently on its own [14, 19–21].

The reference [22] provides an overview of fuzzy logic as well as information on how to develop fuzzy logic algorithms utilizing LabVIEW FPGA. Despite the fact that the reference was published in 2016, the LabVIEW functions referenced seem to require LabVIEW 8.2, which was released 10 years earlier. Moreover, the toolkit subroutines [23] are absent, contrary to what is mentioned in the book [22]. Access to the Toolkit that the authors produced for academic use is expressly reserved by the publisher and/or the authors.

DC motors are extensively utilized in the field of robotics and mechatronics. Typically, these motors are utilized within position servo systems. In various applications, such as those involving mobile robots, the regulation of DC motor velocity is a crucial factor. Several research papers have presented different approaches and configurations for implementing Fuzzy Control in DC Motors. The paper by Garrido et al. [24] introduced a novel approach for regulating the velocity of a DC motor, which involves the utilization of a proportional plus adaptive fuzzy compensation technique. The methodology employed in this study is based on Lyapunov theory and compared with a conventional proportional integral controller. The algorithms were executed in the Matlab–Simulink real-time environment in HIL manner. Sanjay et al. [25] proposed a fuzzy logic controlled robotic system for person follower behaviour in an indoor environment. The robot's movement is governed by a fuzzy logic algorithm that involves the implementation of fuzzy control on the host computer by utilizing the real-time processor in a Hardware-in-the-Loop (HIL) interface. The FPGA is only used for generating the pulse modulation and processing the encoder pulses into speed signals. Several papers employed a similar implementation approach, whereby the controller was activated on the real-time processor, while the FPGA board served as a supplementary component for facilitating communication with the sensors and actuators and interpreting digital signals with lesser computational time [3, 26, 27].

In this paper, FLC-FPGA stand-alone controller is developed, synthesized, simulated and implemented on reconfigurable FPGA board for fast mechatronic systems. Instead of building the membership functions using lookup tables, as mentioned in [23, 28, 29], we developed our set of LabVIEW functions based on logic operations. The advantage and disadvantage of the proposed technique will be discussed throughout this paper. The FLC-FPGA has been design using systematic approach. The results of the FLC implemented on FPGA have been compared with the results obtained using PID-FPGA on LabVIEW Real-Time environment. The obtained results indicate the possibility of successful application of the proposed FLC-FPGA controller for embedded mechatronic systems in which the reference signal changes at a frequency not exceeding $50[Hz]$.

This article provides a concise method to deploy Fuzzy Logic Control (FLC) using RIO-FPGA technology. We also provide a technique for implementing the three stages that constitute the FLC structures in LabVIEW environment using fixed math operations. This paper is organized as follows; in Section 2, we describe the FLC in wording that facilitates the understanding of the programming mechanism and implementation on the FPGA board. Then in Section 3, we explain the features and capabilities of the NI-SbRIO that used

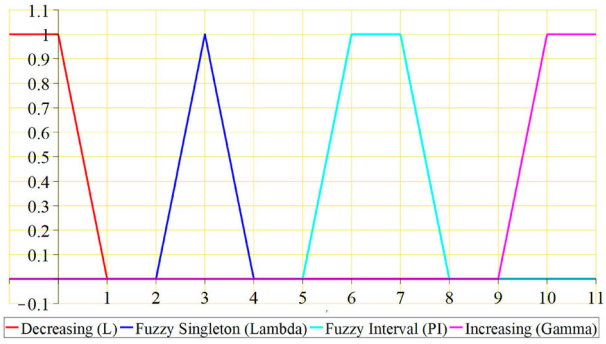

Figure 1. The four basic constructors.

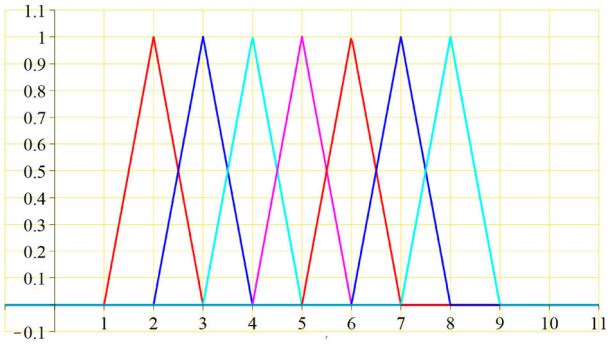

Figure 2. sequence of fuzzy sets.

as the embedded controller, afterwards the programming of the controller and the construction of necessary subroutines (sub-VIs) are presented in Section 4. The next section explains the laboratory verification of the proposed FLC-FPGA of a Sugeno-type fuzzy logic controller with five triangular membership functions for two inputs and with no membership functions for the output. Finally the results and conclusion are summarized.

2. Fuzzy logic control

Fuzzy logic is a multivalued logic with truth represented by a value on the closed interval $[0, 1]$, where 0 is equated with the classical false value and 1 is equated with the classical true value. Values in $(0, 1)$ indicate varying degrees of truth. The first component in the FLC is the fuzzifier that converts crisp inputs into a set of membership values in the interval $[0, 1]$ in the corresponding fuzzy sets. Fuzzy sets are often defined by piecewise functions, which can be defined using predefined constructors. The four basic constructors, L, Γ, Π and Λ are named after the shapes which they construct (L, Γ, Π and Λ, respectively) [2], see Figure (1).

In addition to the default linear model, numerous other forms of fuzzy sets may be constructed by indicating the models to be used. For example, the quadratic model constructs fuzzy sets as defined by Zadeh's S-functions. Additional models are based on crisp sets; cubic polynomials; and rational, exponential, arc-tangent and hyperbolic-tangent functions. An interval can be partitioned into a sequence of fuzzy sets using piecewise functions, as shown in Figure 2.

Fuzzy controllers take input variables (observables) and, based on a set of rules, construct a controlling response. The entire fuzzy logic controller is divided into mainly three different subsystems: fuzzification, fuzzy rule-based decision making and defuzzification as shown in Figure (3). In the fuzzification phase, a linguistic variable requires the conversion of its numerical inputs (crips set) to its normalized form (linguistics). The inputs of FLC, usually the error and error gradient, are normalized using the factors as K_e, $K_{\dot{e}}$ to be in the range of $[-1, 1]$. The Membership Functions (MFs) are selected by the designer based on scaling parameters. Finer control is achieved with a narrow band of MFs near the zero regions. The wider band of MFs away from the zero region results in faster system response [30].

The fuzzy rule table is a matrix of values that defines what the output control surface should look like. The number of inputs and amount of inputs along with the number of fuzzy membership functions will determine the table dimensions. The values that are placed in the rule table indicate the number of membership functions used in the fuzzy system.

The Defuzzification phase, called Inference, utilizes Fuzzy Logic (FL) operators to map the function between input and output. The two basic methods of fuzzy inference are Mamdani and Sugeno. Both Mamdani and Sugeno forms of fuzzy rules may be used [9, 30]. Both zeroth-order and higher-order Sugeno forms can be implemented. The fuzzy inferences used with the Mamdani form of fuzzy rules include both Mamdani and Gödelian combinations. The first includes the minimum and product inferences whereas the second includes Dienes–Rescher, Lukasiewicz and Zadeh inferences. The forms and inferences used may be quickly changed simply by updating options passed to the controller routine. The gain factor K_c for the output is determined in such a way that the output of the FLC can generate the required control action.

Fuzzy systems are traditionally done on a microprocessor. The traditional fuzzy controller contains only two or three inputs that do not contain very many bits. The problem with a fuzzy system is the output control surfaces can be very choppy. This paper proposes an optimized implementation procedure of fuzzy control system using a new FPGA technology with weighted average concept to keep the fuzzy lookup table small. A concise description of the FPGAs will be demonstrated in the following section.

3. Field programmable gate arrays (FPGA)

An FPGA is analogous to a printed circuit board that has a number of unconnected devices on it.

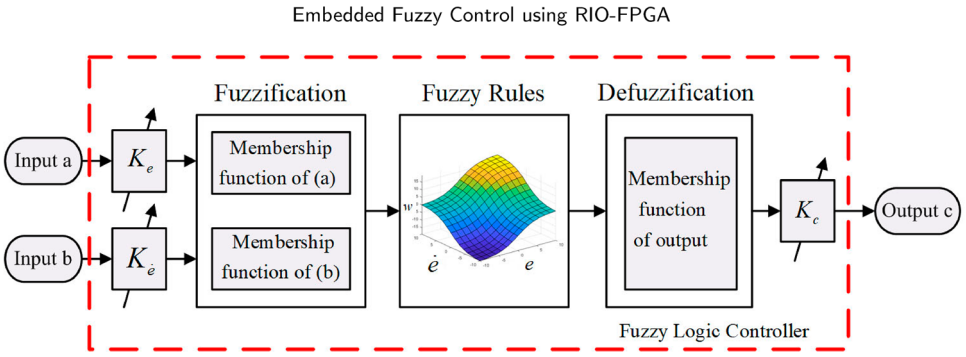

Figure 3. Schematic of Fuzzy Logic Controller.

Traditionally, the devices are connected with physical wires soldered to the pins with a wire wrapping tool or embedded with the printed circuit board. The physical wires are difficult to modify; however, the connection in an FPGA circuit is dynamically defined by software programming. The program causes semiconductor switches to turn ON or OFF, thereby defining the connections between gates. Therefore, an FPGA can be defined as a programmable chip composed of three basic components: logic blocks, programmable interconnects and Input/Output blocks [11, 13]. A single FPGA chip can replace thousands of discrete components by incorporating millions of logic gates. Figure 4 shows an FPGA as a reconfigurable digital architecture with a matrix of configurable logic blocks with horizontal and vertical routing channels surrounded by a periphery of I/O blocks. Signals can be routed within the FPGA matrix in any arbitrary manner by programmable interconnect switches and wire routes. Traditionally, programming these FPGA chips is difficult and, therefore, it is only allowed by experienced digital designers and hardware engineers. National Instruments (NI) has simplified programming of these devices via visual programming system design with LabVIEW FPGA-module, so that the advantages of these powerful reconfigurable chips can be utilized [31].

First, one can use the LabVIEW FPGA module to define the FPGA logic, instead of using the low-level language such as very high speed integrated circuit hardware description language (VHDL). LabVIEW FPGA generates the VHDL code and passes it to the Xilinx[1] compiler. Then the Xilinx compiler synthesizes the VHDL and routes all synthesized components into a bitfile. The compiled bitfile is downloaded to the FPGA chip. FPGA logic provides timing, triggering, processing and custom I/O measurements. One can implement multiloop analog control systems at loop rates exceeding 100 KS/s, and digital control systems at loop rates up to 1 MS/s, and it is possible to evaluate multiple rungs of Boolean logic using single-cycle While-Loops at 40 MHz. LabVIEW and FPGA can implement parallel tasks in hardware to process and generate synchronized analog and digital signals

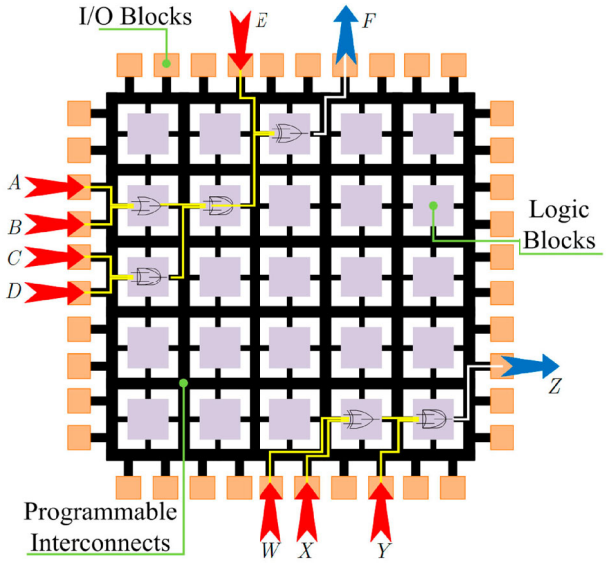

Figure 4. Reconfigurable FPGA chip.

rapidly and deterministically. FPGA module creates dedicated hardware for each independent function in FPGA VIs. Furthermore, the FPGA module does not have an operating system that can allocate the time of the Central Processing Unit (CPU) to various activities in a task-based manner. In addition because of the parallel architecture of the Reconfigurable-Input–Output (RIO) core, increasing computational power does not always result in a decrease in FPGA application speed.

To attain parallel processing, one should separate the code into different and independent segments. In mechatronic systems, for example, self-governing loops can be programmed to acquire and quickly process measured analogue or digital data at distinct loop rates.

In the experimental validation of this paper, single board RIO (sbRIO), shown in Figure (5), is employed. The sbRIO-9631 in this instance combines a 110 digital I/O line, 1 M gate Xilinx Spartan FPGA and a 266 MHz real-time processor. It offers 128*MB* of nonvolatile memory for storing programs and data logging along with 64 MB of DRAM for embedded operation. This device has an integrated 10/100 Mbits/s Ethernet connection that allows for network signal communication. The real-time processor plays an important role in

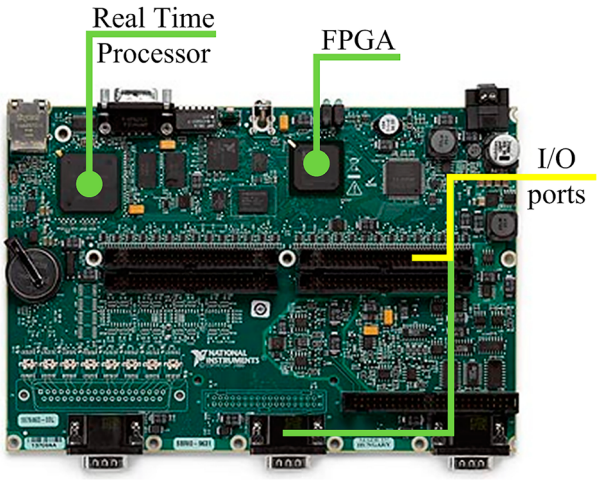

Figure 5. sbRIO-9631 www.ni.com.

the Hardware In Loop (HIL) applications. The design flow and graphic outlines of FPGA design that begins with evaluating the system and ends with deploying the code are clarified in [32].

An actuator (DC motor), sensor (Encoder) and drivers (linear/pulses) are the three major parts that construct up a preliminary mechatronic system. The FPGA layout should have two basic independent loops for the purpose of operating and controlling the mechatronic system: one for measuring and feeding back the angular velocity via the optical quadrature encoders and another for generating the required pulse width modulation (PWM) to drive the actuator. Along with these fundamental loops, additional loops must be incorporated together during the implementation process. One loop is used to calculate the control law, and another loop is used to pass the collected data to the real-time processor for the validating process. Figure 6 shows the layout of the parallel processing loops. The details of measuring and driving loops are found in [12, 14, 33]. Whereas the contribution of this paper is concerned in the calculation of the fuzzy logic control law within the FPGA environment, referred to as FLC-FPGA, which will be discussed in the following section.

4. FLC-FPGA

FPGA module restricts the use of mathematical operations in FPGA VIs to integer numeric data types and it is not possible to use the floating point operations. When integer math is used, the results may overflow. Integer overflow occurs when the result of a mathematical operation exceeds the range of the output data type. To avoid the integer overflow, one can use the scale by power of 2 to reduce the magnitude of the inputs or to use a larger output data type. The scaling minimizes the amount of space used in FPGA device, but on the other side, loses the precision of the mathematical operations. Larger output data type takes more space on FPGA device but performs the processing more quickly and receives more accurate data. Furthermore, only fixed-size, one-dimensional arrays in FPGA VIs can be used. The arithmetic manipulation that returns a variable-size array cannot be used. Since arrays consume significant amounts of space on FPGA, therefore, arrays larger than 32 elements should be avoided.

In this study, a Sugeno-type fuzzy logic controller with five triangular membership functions for two inputs and with no membership functions for the output is examined for implementation in embedded mechatronic systems.

4.1. FPGA – fuzzification

A membership function (MF) is a function that associates each point in X, where $x \in X$, with a real integer in the range $[0, 1]$. There are several MF which can be used as Fuzzy sets, as described earlier, that can be constructed using the basic four constructors, L, Γ, Λ and Π. The mathematical representation for triangular MF (Λ), see Figure (7), is defined by its parameters $\{a, b, c\}$ such that

$$\mu_A(x) = \begin{cases} 0 & x \le a \\ \frac{x-a}{b-a} & a \le x \le b \\ \frac{c-x}{c-b} & b \le x \le c \\ 0 & x \ge c \end{cases} \tag{1}$$

The parameters $\{a, b, c\}$ with $a < b < c$ determine the x-coordinates of the three corners of the underlying triangular MF. Triangular MFs can be asymmetric. Depending on the relationships between a, b and c, triangular MFs may be asymmetric. A trapezoidal MF (Π) is specified by four parameters $\{a, b, c, d\}$, see Figure (8), is defined as

$$\mu_A(x) = \begin{cases} \frac{x-a}{b-a} & a \le x \le b \\ 1 & b \le x \le c \\ \frac{d-x}{d-c} & c \le x \le d \\ 0 & x \le a \cup x \ge d. \end{cases} \tag{2}$$

The Crisp is defined by its parameter c such as

$$\mu_A(x) = \begin{cases} 1 & x = c \\ 0 & otherwise. \end{cases} \tag{3}$$

Instead of building the membership functions using lookup tables, as mentioned in [23, 28, 29], we developed our set of LabVIEW functions based on logic operations and fixing some geometrical parameters that do not affect the subsequent control operations. Fixing the distances, for instance, between the triangle MF's left foot, peak, and right foot and controlling the trapezoidal MF's top points, left and right shoulders, and other characteristics.

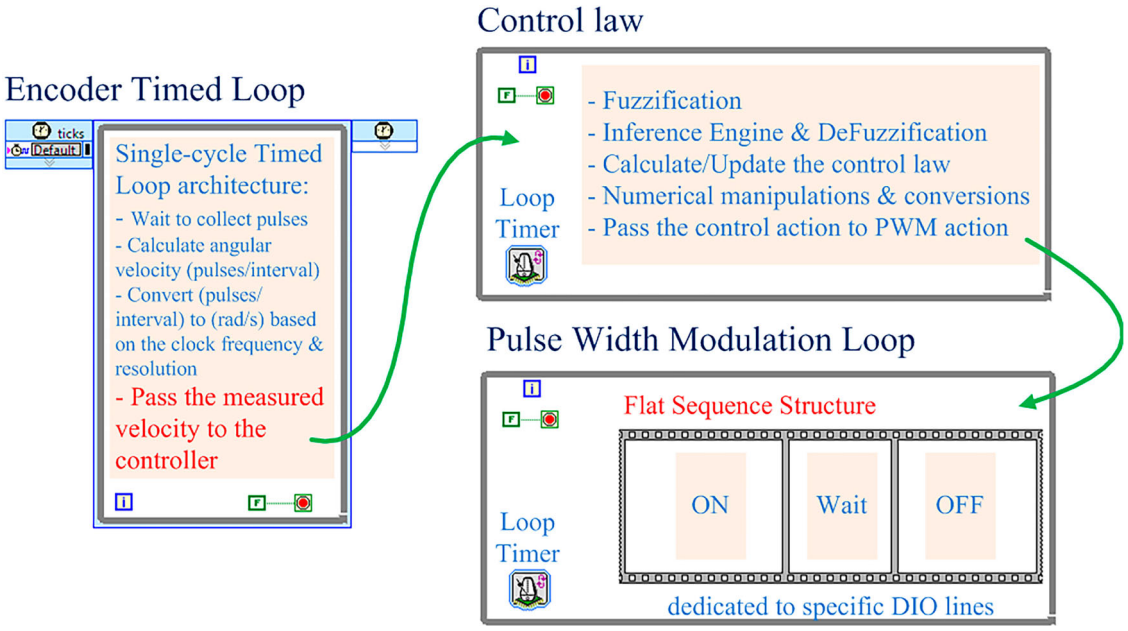

Figure 6. Preliminary FPGA layout with measuring, driving and control loops.

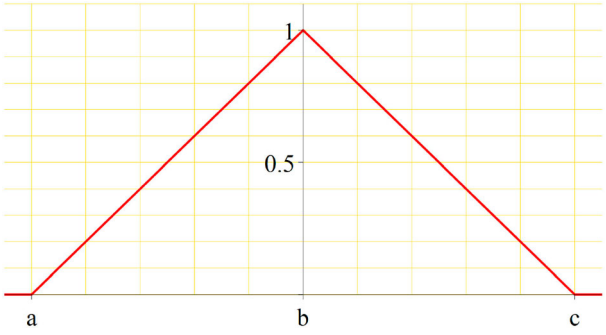

Figure 7. Triangular MF.

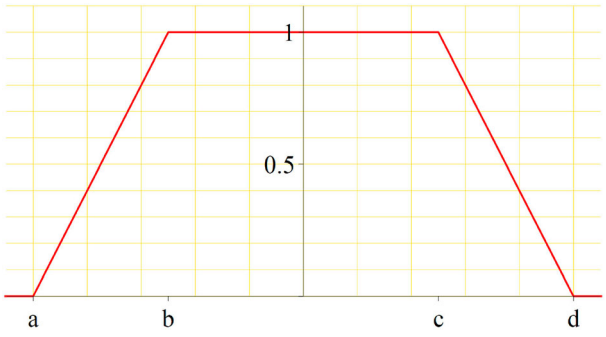

Figure 8. Trapezoidal MF.

Using max–min operators, Equation (1) can be represented as

$$\mu_A(x) \equiv A(x) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right). \tag{4}$$

In a similar way, Equation (2) can be written as

$$\mu_A(x) \equiv A(x) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right). \tag{5}$$

Due to the fact that the normal divide function is not supported in LabVIEW FPGA and the quotient and remainder function imposes a constant delay, Equations (4) and (5) can be modified, without loss of generality, into the following forms:

$$A(x) = \max(\min(2(x-a), 2(c-x)), 0) \Leftrightarrow b-a = c-b = 0.5 \tag{6}$$

$$A(x) = \max(\min(4(x-a), 1, 4(d-x)), 0) \Leftrightarrow b-a = d-c = 0.25. \tag{7}$$

Left and Right triangle MFs can be constructed by setting $a = b$ and $c = b$ respectively. Using Equations (6) and (7), the membership values of the input/output variable can be obtained. It should be mentioned that the results are almost as accurate as those from a double precision version written in LabVIEW. Given an input value x, the membership value obtained is a floating-point value in a range between 0 and 1. Since all variables are handled in integer format, this value is challenging to implement in hardware and also uses additional system resources. Consequently, the values are scaled to integers based on the needed resolution in accordance with

$$A_{Scaled} = A \times \left(2^n - 1\right). \tag{8}$$

In this manner, floating point values are scaled to a $(2^n - 1)$ resolution, where n represents the number of bits used in the system. To make the calculations easier, the probability scale utilized in this study is suggested to be 10, 000 , equivalent to n equals 14, that is the saturation limit for the computation.

In LabVIEW FPGA module, the triangle membership function can be established through the programming of Equations (6) and (8), as presented in Figure 9.

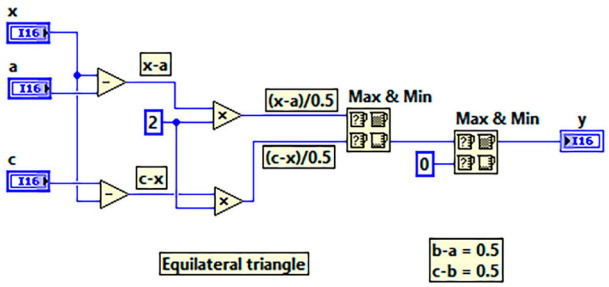

Figure 9. LabVIEW *TriangleMF.vi*.

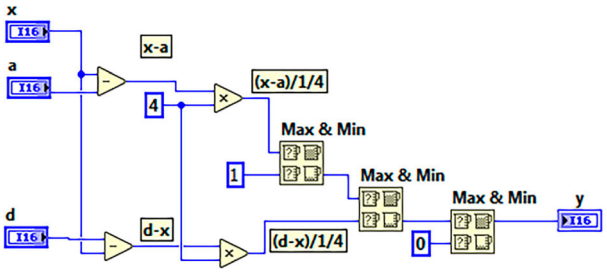

Figure 10. LabVIEW *TrapezoidalMF.vi*.

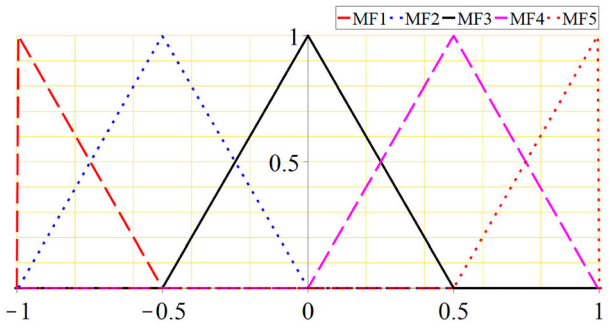

Figure 11. Fuzzy sets of the error variable.

Also, the trapezoidal membership function is presented in Figure 10. Note that the Max & Min Function compares *two* inputs and returns the larger value at the top output terminal and the smaller value at the bottom output terminal.

Consequently, five triangular membership functions subjected to input's variable can be constructed as shown in Figure 11. Each membership MF_1 to MF_5 describes the probability of the input variable as $(-\Downarrow)$ Negative Big, $(-\downarrow)$ Negative Small, $(\longleftrightarrow)$ Zero, $(+\uparrow)$ Positive Small, and $(+\Uparrow)$ Positive Big. The Labview programming of FPGA-Fuzzification process is constructed via LabVIEW sub-VIs called 5*MFTri.vi* by using five *TriangleMF.vi* with the scaled boundaries from $-10,000$ to $10,000$, see Figure 12. The output of the 5*MFTri.vi* is a fixed-size array of 5 elements represents the probability of the input conditions. For example, if the input value is -0.3 (-3000 scaled value), then the array of input probabilities should have the form of:

$$A_{Scaled} = \begin{bmatrix} 0 & 6000 & 4000 & 0 & 0 \end{bmatrix}^T. \qquad (9)$$

In the case of manipulating more than one input variable, i.e. two inputs, such as the error e and the

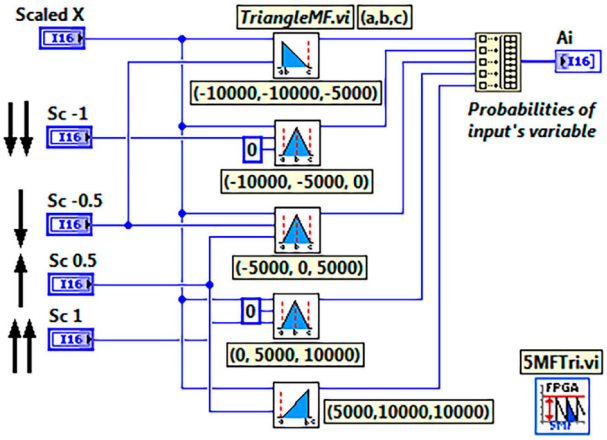

Figure 12. Input error fuzzy sets – 5*MFTri.vi*.

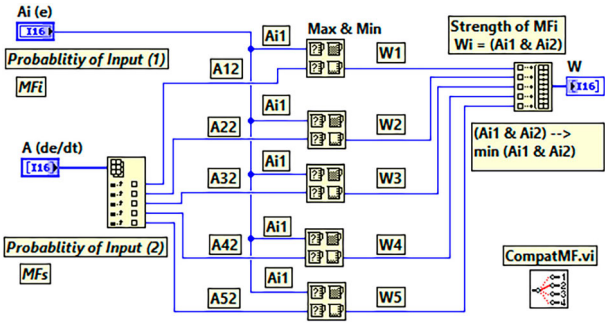

Figure 13. Strength of inputs' MFs – *CompatMF.vi*.

change of error $\dot{e}$, one can estimate the measure of the influence of membership functions. This can be done by estimating the strength of each rule, w^i, according to the corresponding probabilities, as follows:

$$w^i = A_1^i \cap A_2^i, \qquad (10)$$

where $A_{1,2}^i$ is the probability of membership function i for inputs 1 and 2 respectively. The fuzzy OR/AND operator simply selects the maximum/minimum of the two probabilities. Figure (13) shows the LabVIEW block diagram of *CompactMF.vi* that estimates Equation (10) using the AND operator by utilizing the Max & Min Function. Note that the lower terminal denotes the minimum value. The output of *CompactMF.vi* is a vector includes the strength of the rules of one probability of input 1, with all probabilities of the input 2.

4.2. FPGA – inference engine and defuzzification

In the rule base unit, there are a set of rules that relate to input and output variables of the controller. These rules are simple if–then structures with a condition and conclusion. For example, rule 1: if Error e is $(-\Downarrow)$ & Change of error $\dot{e}$ is $(-\Downarrow)$ then the appropriate value from the rule table c^1 is -1, where $(-\Downarrow)$ means Negative Big. Table 1 shows the appropriate value of each

Table 1. Rules Table: $(-\Downarrow)$ Negative Big, $(-\downarrow)$ Negative Small, $(\longleftrightarrow)$ Zero, $(+\uparrow)$ Positive Small, $(+\Uparrow)$ Positive Big.

$e/\dot{e}$	$-\Downarrow$	$-\downarrow$	$\leftrightarrow$	$+\uparrow$	$+\Uparrow$
$-\Downarrow$	−1	−1	−1	−1/2	0
$-\downarrow$	−1	−1	−1/2	0	1/2
$\leftrightarrow$	−1	−1/2	0	1/2	1
$+\uparrow$	−1/2	0	1/2	1	1
$+\Uparrow$	0	+1/2	1	1	1

rule, i.e. $c^i s$, where $i = 1 \longrightarrow N$ and N is the number of rules.

The inference engine manages the data and the rule base and it decides that the appropriate values are used. There are two types of inference systems vary somewhat in the way outputs are determined. Mamdani's fuzzy inference method, the first control systems built using the fuzzy set theory, is the most commonly seen fuzzy methodology. Mamdani-type inference expects the output membership functions to be fuzzy sets that needed defuzzification. However, it is possible to use a single spike as the output membership function rather than a distributed fuzzy set. This type of output is known as a singleton output membership function. It enhances the efficiency of the defuzzification process because it greatly simplifies the computation required by the more general Mamdani method. Instead of integrating across the two-dimensional function to find the centroid, one can use the weighted average of a few data points to calculate the control output. Sugeno-type systems support this type of model. A typical rule in a Sugeno fuzzy model for two-input single-output has the form: If Input 1 is x and Input 2 is y, then Output is $z = ax + by + c$, and for zero-order Sugeno model, the output level z is a constant ($a = b = 0$), i.e. $z = c$. Thus the final output of the system is the weighted average of all rule outputs, computed as

$$Final\ Output = \frac{\sum_{i=1}^{N} w^i z^i}{\sum_{i=1}^{N} w^i} \tag{11}$$

where w^i is computed by using Equation (10) and N is the number of rules according to Table 1. The weighted average method is suitable for symmetric membership functions, see Figure (11).

It is concluded that three stages of calculations are required to construct the FPGA-Fuzzy Logic controller on LabVIEW. First, in the fuzzification stage, the two inputs are checked to obey the saturation limits ($\pm 10,000$) and then introduced, individually, to two *5MFTri.vi* to generate five probabilities of the each input (error and the change of error). Each probability output (one value per membership) of the input 1, i.e. error signal, is fed-forward to *CompactMF.vi* with all probabilities of the input 2 (change of error), to generate the strength of each pair corresponding to the rule table.

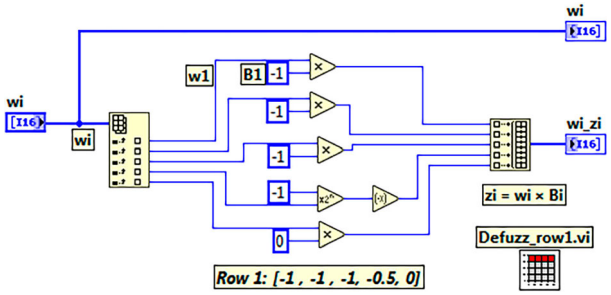

Figure 14. First row of 0-order Sugeno model *Defuzz—row*1*.vi*.

For example, the probability of the $MF1$ of the error signal, $A_i(e)$ in the block of Figure 13, with the probabilities of $MF1$ to $MF5$ of the change of error, $A(\mathrm{d}e/\mathrm{d}t)$, generates the strength vector corresponding to first row of the rule table, Table 1, and therefore the weighted output can be estimated using the block diagram shown in Figure 14.

Figure 15 shows the complete block diagram of the three stages of the FLC-FPGA controller proposed in this paper. In the second stage, step 1 involves five calls of *CompactMF.vi* are carried out to generate the strength vector of each row of the rule table, followed by step 2 that estimates the weighted outputs of each row according to zero-order Sugeno model. The Defuzzification stage estimates the final output according to Equation (11). Note that the definition of the outputs has no membership functions, by doing this the computation of the centroid of mass or any other defuzzification method is avoided because there is only one value defined for the output.

4.3. Fuzzy controller

As described earlier, in the FLC, it is assumed that the mathematical model of the system is unavailable; instead, only the system output, y, can be monitored and the system's states, particularly; the error can be estimated and change of error can be approximated. Depending on the type of control type required, i.e. PD, PI, or PID type. The control law of the fuzzy controller can be estimated using the error, $e(k)$, change of error $\Delta e(k)$, and the sum of error $\sum e(k)$ as inputs and control input $u(k)$ as output, where k is the sampling time. Thus typical proportional differential PD-like FLC can be developed by [34, 35]

$$u(k) = K_p.e(k) + K_d.\Delta e(k), \tag{12}$$

where K_p and K_d are the proportional and differential gain coefficients. The error and change of error are defined as

$$e(k) = y_r - y(k) \tag{13}$$

$$\Delta e(k) = e(k) - e(k-1), \tag{14}$$

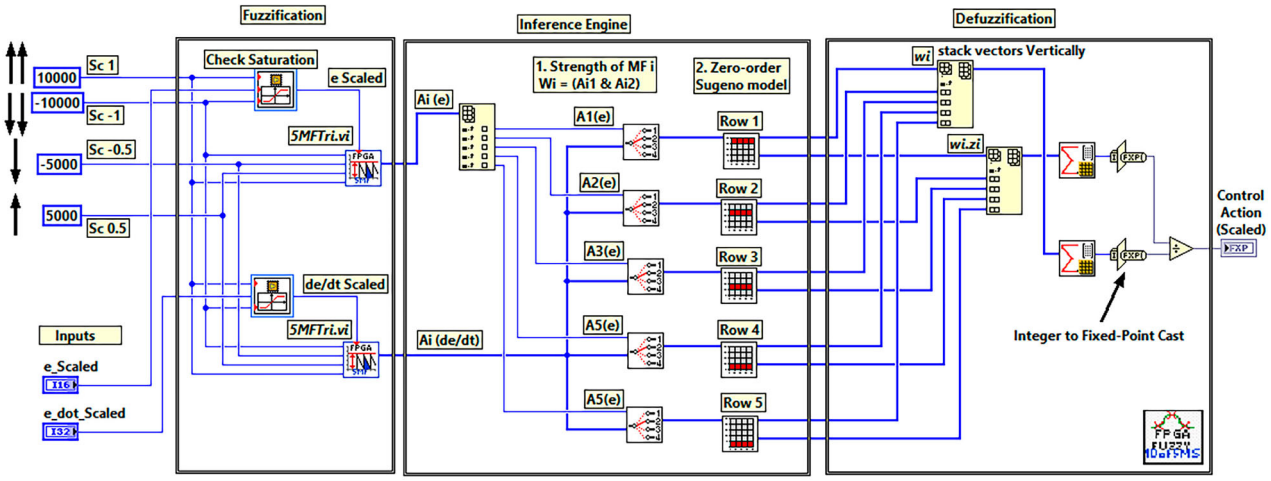

Figure 15. LabVIEW Block diagram of FLC-FPGA, FLC—FPGA—5mf—2I— 1O.vi.

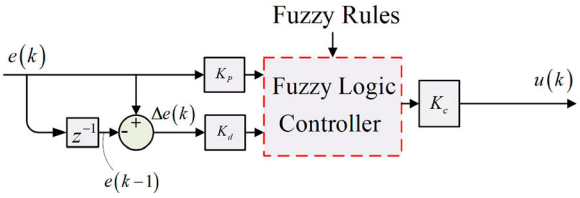

Figure 16. Block diagram of PD-type fuzzy controller.

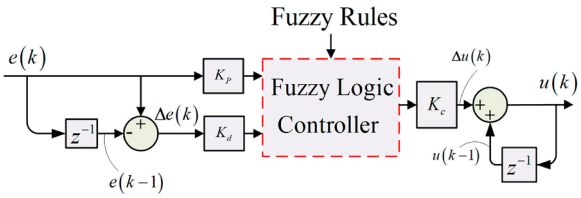

Figure 17. Block diagram of PI-type fuzzy controller.

where y_r is the reference input, i.e. desired output and $y(k)$ is the real output. Figure 16 shows the block diagram of a PD-FLC with error and change of error as inputs. The inverse $z-$ operator, z^{-1}, represents the unit time delay. The error $e(k)$ and the change of error $\Delta e(k)$ are fed into the FLC. The error and change in error are first normalized by using two gains K_p and K_d , respectively. These normalized gains are fed to the FLC and after a process of defuzzification, the FLC gives a control signal which has a value within the pre-described range of discourse. The control action is then obtained by multiplying the control signal by an additional constant, K_c, known as the output scaling gain.

The classical PI control action in time domain takes the form of $u(t) = K_p.e(t) + K_I.\int e(t)\,\mathrm{d}t$, differentiating both sides, yields

$$\frac{\mathrm{d}}{\mathrm{d}t}u\,(t) = K_p.\frac{\mathrm{d}}{\mathrm{d}t}e\,(t) + K_I.e\,(t)\,. \tag{15}$$

Using the forward differential approximation, the discrete form of Equation (15) takes the form of

$$\left(1 - z^{-1}\right)u\,(z) = K_p.\left(1 - z^{-1}\right)e\,(z) + K_I.e\,(z)$$

Taking the inverse $z-$ transform, yields

$$u(k) - u(k-1) = K_p\,(e\,(k) - e\,(k-1)) + K_I e\,(k) \tag{16}$$

$$\Updownarrow$$

$$\Delta u(k) = K_p \Delta e(k) + K_I e\,(k) \tag{17}$$

Equation (17) is similar to Equation (12), however, the output is the difference of the control action and the integral constant is multiplied by the error signal while the proportional constant is multiplied by the error difference. Figure 17, which depicts the structure of the PI-FLC controller, shows the accumulation of the output of the PD-FLC, to produce the FLC with the PI effect, note that the output of the first stage is $\Delta u(k)$. Using a digital approximation for integration, the control signal $u(k)$ is obtained as

$$u(k) = \Delta u(k) + u(k-1). \tag{18}$$

A fuzzy controller structure that resembles a PID-like controller can be created by combining PD-FLC with PI-FLC control actions. Figure 18 depicts the FLC-PID control system's entire structure. The output of the PID-FLC is the sum of the outputs of the PD-FLC and PI-FLC, and it has two scaling gains, (K_c^{PD}, K_c^{PI}), for the FLCs of types PI and PD, respectively,

$$u(k)|^{PID-FLC} = K_p.e\,(k) + K_d.\Delta e\,(k) + \Delta u(k) + u(k-1). \tag{19}$$

Given the output of the defuzzification phase, see Figure 15, Equation (18) will be used to add the integral part to the final control action, that is expressed by Equation (19).

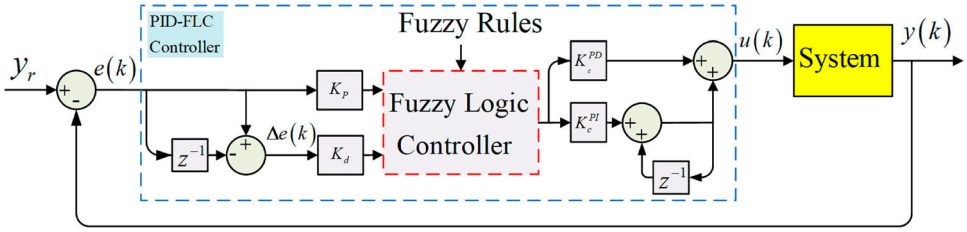

Figure 18. Block diagram PID-FLC control system.

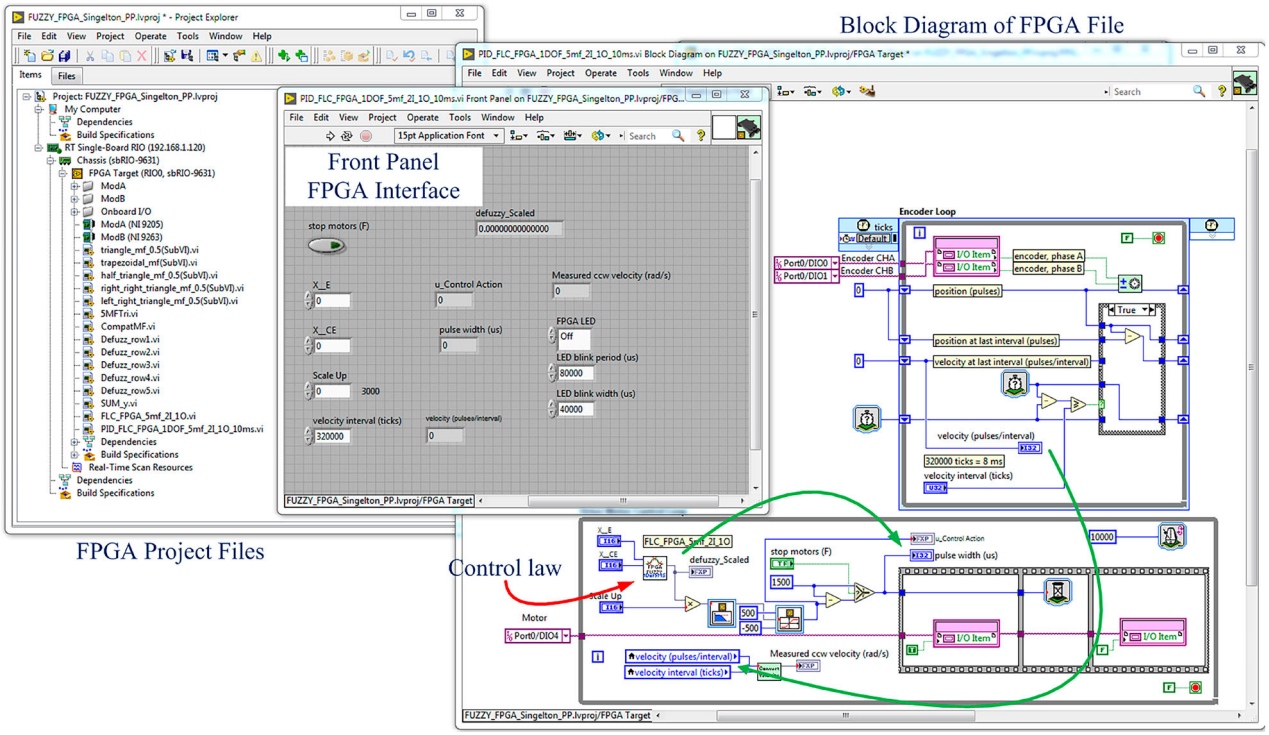

Figure 19. LabVIEW FPGA project.

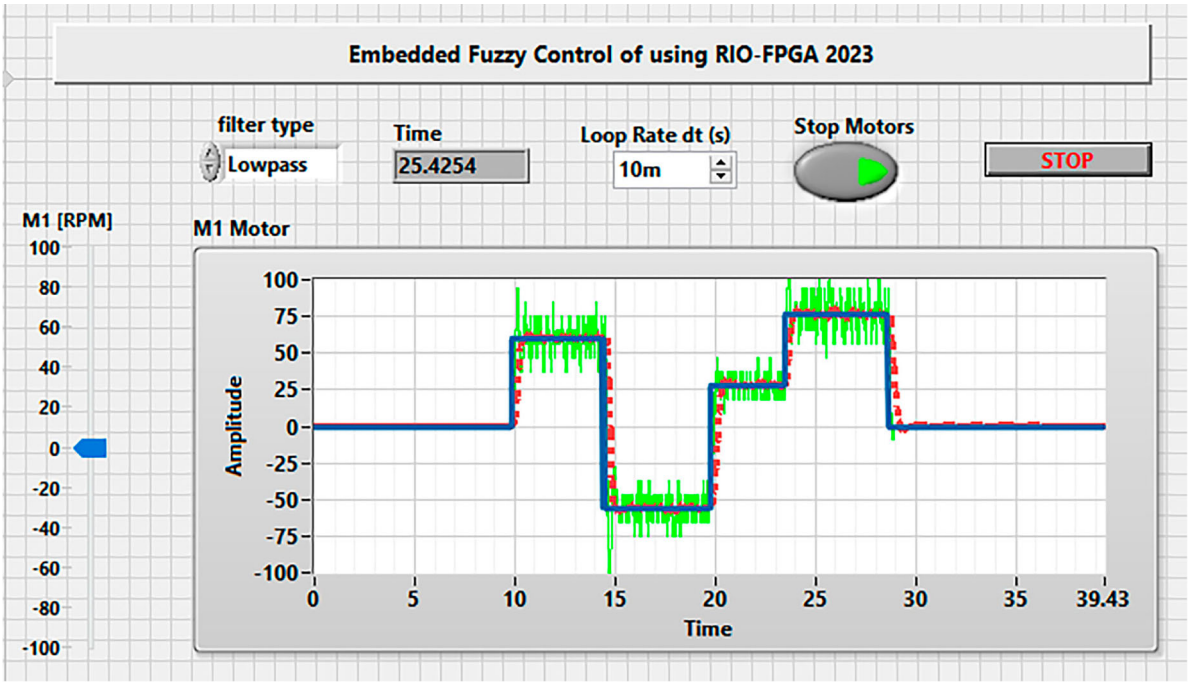

Figure 20. LabVIEW front panel interface with the FPGA target.

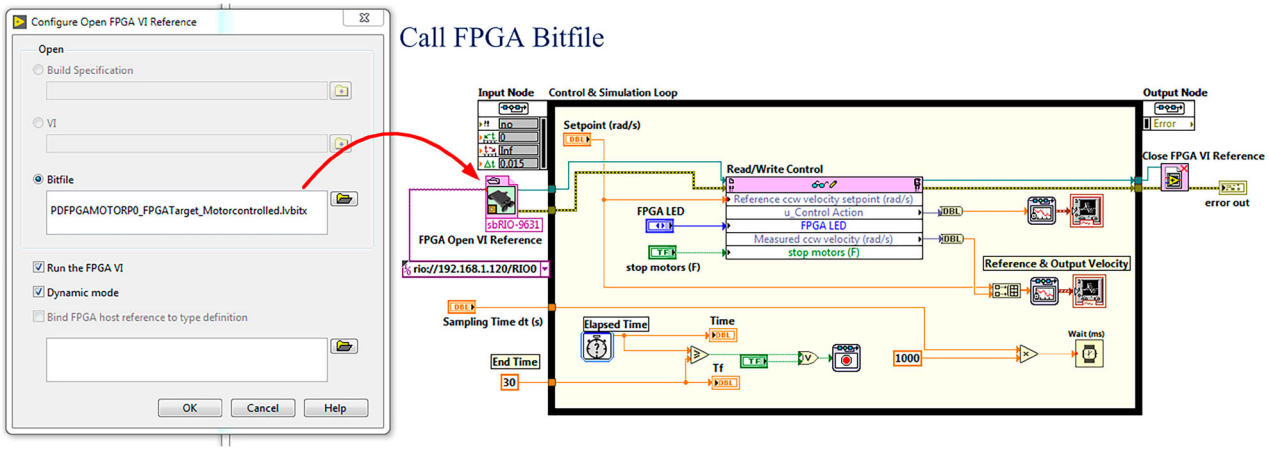

Figure 21. Block diagram of the interface with FPGA target shows the Bitfile call.

4.4. LabVIEW implementation of embedded PID-FLC-FPGA controller

FPGA applications can range from a small embedded system with a single FPGA target, one or more RT targets, and a development computer to a large embedded system with several FPGA targets. One can use LabVIEW project files (.lvproj) to manage targets and VIs, references to project files, configuration data; deployment data; build data and other data, see Figure 19. This project includes the following items: The sbRIO-9631 chassis houses and directly connects the I/O blocks of the FPGA target (ModA, ModB and Inboard I/O) to the interchangeable I/O modules for high-performance timing, triggering and synchronization. The controller (RT Single-Board RIO) attaches instantly to the sbRIO-9631 chassis and communicates either directly or via a network with the development computer. The FPGA VIs (subroutines of the FLC in the previous section) are the VIs that would be downloaded and run over the FPGA target. The LabVIEW compilation tools translate the FPGA VI into a circuit schematic to reconfigure the blocks and interconnect on the FPGA target. Figure 19 shows the FPGA project files as well as the main FLC-FPGA control file, the VI is called as *PID_FLC_FPGA_1DOF_5mf_2I_1O_10ms.vi* that call all the above VIs with loop rate of 10[*ms*]. In last, project files are configured and translated throughout the compilation process into a bitfile that can be downloaded to the FPGA target.

5. Experimental work

This paper compares every result with traditional PID control of an actual robot platform, a National Instruments robot platform (DaNI 1.0). A DaNI robot comes preassembled and has two DC motors for each side, installed on the front wheels with encoders and the Starter Kit 1.0 (sbRIO) VIs are included in the package. It is equipped with the sbRIO-9631 embedded control board, integrating a user-reconfigurable field-programmable gate array, as presented earlier. The sbRIO-9631 can be programmed in the LabVIEW graphical development environment. The onboard reconfigurable FPGA can be quickly programmed using the LabVIEW FPGA Module for high-speed control. LabVIEW contains the necessary built-in drivers for handling data transfer between the FPGA and real-time processor.

The Starter Kit 1.0 (sbRIO) VIs includes a ready-made classical PID control employed in the FPGA environment and referred to in this investigation as PID-FPGA. The PID-FPGA controller compares the feedback signal with the set point (desired angular velocity) and manipulates the error signal to generate a pulse width modulated signal (PWM) as a control signal. The frequency of the PWM signal takes a value from 1000−2000 [μs], 1000 [μs] for full-speed in counter-clockwise direction and 2000 [μs] for the full-speed of the motor in the clock-wise direction, and 1500 [μs] for zero rotation. The percentage of duty cycle of the PWM signal is varied by the PID-FPGA controller, which switches the driver ON/OFF to deliver the necessary power to the motor so as to achieve the desired angular velocity of the DC motor. The velocity is fed back to the controller through a two-channel optical encoders that are attached to the motor and produce 400 pulses per revolution. Similar PID-FPGA control structures have been developed in previous research work [14, 36] and for different applications with acceptable and encouraging results. In this section, the obtained results of the PD-FLC and PID-FLC -FPGA controllers developed in this work will be compared to the classical PID-FPGA.

Given that the FPGA code developed in this paper that utilizes the membership functions depicted in Figures 7 and 8 and simplified so that the memberships are vertically symmetrical, this may prohibit the membership functions from being optimized. However, it is possible to optimize the controller by adjusting the rules of the inference system as well as the scaling constants. Rules of Table 1 may serve as an initial selection

and can be further refined (optimized) through using the utilization of the genetic algorithm that minimize a predetermined cost function [37].

By employing Figure 18 as a fundamental framework for implementing Fuzzy-FPGA control upon an embedded system, it is possible to outline the stability boundaries of scaling constants. The proportional, integral and derivative constants can be described as [38, 39]

$$\left.\begin{aligned}\mathbf{K_P} &= K_c^{PD}\cdot K_p + K_c^{PI}\cdot K_d\\ \mathbf{K_I} &= K_c^{PI}\cdot K_p\\ \mathbf{K_D} &= K_c^{PD}\cdot K_d\end{aligned}\right\}$$

where K_p and K_d are the input scaling constants subjected to the error and change of error, while K_c^{PD} and K_c^{PI} are the output scaling constants related to the PD and PI control actions respectively. In the mechatronics motion systems, the direct current motors can be modelled as, neglecting the mechanical friction, first order of time constant $\tau$with an integrator. In this case, the sufficient conditions for the stability are [38]

$$\left(\frac{K_c^{PD}}{\tau} - K_c^{PI}\right)K_p + \frac{K_c^{PI}}{\tau}K_d \geq 0, \qquad (20)$$

which implies that

$$K_c^{PI}\cdot K_p \geq 0, \quad K_c^{PD}\cdot K_d \geq 0, \quad \frac{K_c^{PD}}{\tau}K_p \geq K_c^{PI}K_p. \qquad (21)$$

Under these conditions, the preliminary selection of scaling constants can be performed so as to maintain system stability.

5.1. One degree of freedom controller

To assess the performance of the designed FLC-FPGA controller, step references with two different desired velocity levels, and consequently, different levels of power demand, have been introduced to a loaded DC-motor system. In this section, the tests were performed on one motor side of the DaNi robot among the motors of the robot used. Figures 20 and 21 show the LabVIEW interface, front panel as well as the block diagram, with the FPGA target via a development computer over a network. In this instance, the controller is referred to as a 1-degree-of-freedom controller, and the control action is calculated once for each loop. The identification of the time constant of the motor yielded that being equal to $\tau = 0.068865[s]$ and thus the preliminary selection of the scaling constants can be taken as $K_c^{PD} = 1$, $K_c^{PI} = 0.06$, $K_p = 1$, $K_d = 0.02$ which satisfy the stability conditions of Equation (21).

Figures 22 and 23 show the results as set desired velocity levels are: 10 and $-10[rad/s]$. Figure 22 represents the performance of the various selected controllers that are PD-FLC-FPGA, PID-FLC-FPGA and classical PID-FPGA, with respect to the step reference.

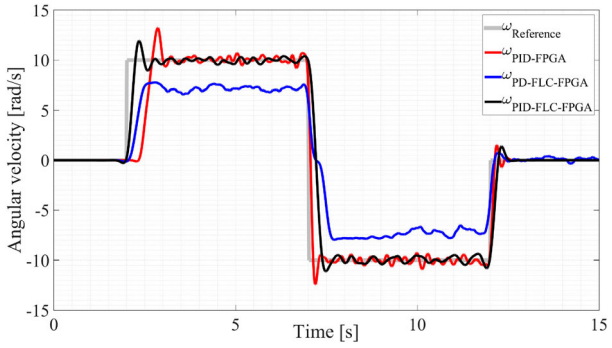

Figure 22. Step response of DC motor.

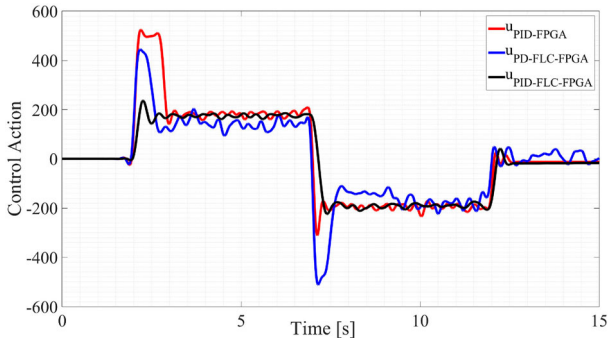

Figure 23. Width modulation of the control action.

It is worth noting that these PD-and PID-FLC-FPGA response curves compared to the traditional PID control, in itself, have a positive contribution. In which, we ensured the success of building FLC-FPGA control structure on a reconfigurable digital environment.

According to this figure, PID has been able to reach the reference velocity at various levels; however, the PD-FLC controller is not. The PD-FLC controller shows a significant amount of steady-state error. This is due to the absence of the integral action on its structure. Furthermore, the PID-FLC-FPGA controller demonstrates a very quick response and little overshoot during the start-up of the system with zero steady-state error. Moreover, introduce a control action that does not exceed the saturation limit of the motor's driver, see Figure 23, where the saturation limits are $\pm 500[\mu s]$.

5.1.1. Optimization of FLC-FPGA controller

Although the primary emphasis of the paper pertains to developing of the Fuzzy controller on FPGA target and not necessarily on the optimization of the controller itself, it is recommended to take into account the effects of optimized variables when undertaking comparative evaluations. In this context, the study has established a cost function that can be expressed as the addition of the overshoot and the root mean square error (RMSE). One may conduct trials using various rule bases and compute the corresponding cost function for each iteration. If the modified set of directives yields better outcomes, it will be retained and employed

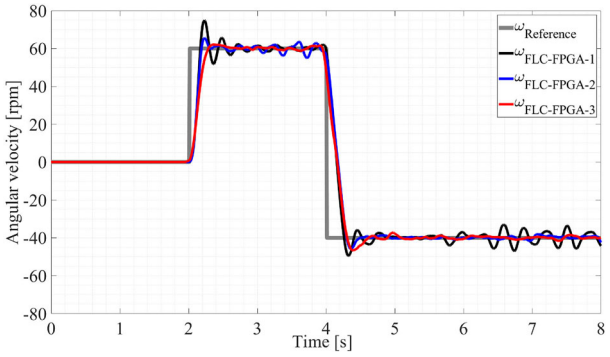

Figure 24. Comparison of optimized versions.

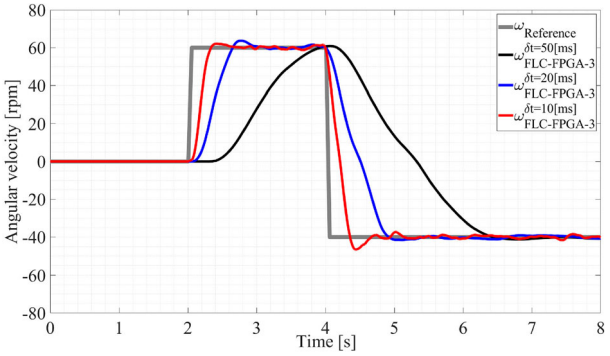

Figure 25. Comparison of FLC-FPGA with various loop rates.

Table 2. Optimized rules table.

$e/\dot{e}$	$-\Downarrow$	$-\downarrow$	$\leftrightarrow$	$+\uparrow$	$+\Uparrow$
$-\Downarrow$	−1	−1	−1/2	−1/2	0
$-\downarrow$	−1	−1/2	−1	0	1/2
$\leftrightarrow$	−1/2	−1/2	0	1/2	1/2
$+\uparrow$	−1/2	0	1	1/2	1
$+\Uparrow$	0	1/2	1/2	1	1

in subsequent attempts to ascertain the optimal solution based on permissible standards. Specifically, the cost function is defined as

$$J = \min\left(|e\,(k)|\right) + \sqrt{\frac{1}{N}\sum_{k=1}^{N} e\,(k)^2} \qquad (22)$$

The MATLAB software (*tunefis*) tool is utilized for the purpose of optimizing rules, employing the genetic algorithm as the tuning methodology. The optimized rules are shown in Table 2 with scaling constants are obtained as $K_c^{PD} = 1.0667$, $K_c^{PI} = 1.6956$, $K_p = 1$, $K_d = 0.0010$. Figure 24 depicts a comparison between the preliminary design of the controller ($FLC{-}FPGA{-}1$), and subsequent updates that involve optimization of rules ($FLC{-}FPGA{-}2$), as well as the optimized rules and scaling constants ($FLC{-}FPGA{-}3$). The comparison illustrates the anticipated improvement in the scenario of optimizing the rules and the scaling constants with regard to the amount of error and the percentage overshoot.

5.1.2. Execution-time calculation and hardware resources

The VIs code is mapped to gates and slices on an FPGA-based target, allowing concurrent loops to be realized on separate regions of the FPGA fabric. This enables all computations to perform concurrently (in parallel). Each loop's time is independent of the other loops, with the flexibility to add operations that let loops to interact and exchange data. To measure the execution time of the proposed controller, different timing VIs can be implemented in parallel on the FPGA without affecting the code's performance. Defining the execution time as the difference between consecutive loop iterations, the execution time of different VIs is shown

Table 3. Execution time of FLC's VIs.

VIs	*Fig./Eq.*	*time*	*Regis*	*LUTs*
5MFTri.vi	Figure 12	$2\mu s$	2210	–
CompatMF.vi	Figure 13	$0.5\mu s$	425	414
Defuzzrow1.vi	Figure 14	$1.6\mu s$	2415	1311
FuzzyFPGA $\cdots$	Figure 15	$6\mu s$	3175	2371
SUM_y	Equation (11)	0	342	–
Encoder	Figure 6	$8000\mu s$	1031	–
Min. Loop rate	Figure 19	$8000\mu s$	–	–

Table 4. Execution time of FLC's using look-up tables.

VIs	*Fig./Eq.*	*time*	*Regis*	*LUTs*
5MFTri.vi	Figure 12	$11750\mu s$	110	5093
CompatMF.vi	Figure 13	$0.5\mu s$	425	414
Defuzzrow1.vi	Figure 14	$1.6\mu s$	2415	1311
FuzzyFPGA $\cdots$	Figure 15	$12000\mu s$	3175	2371
SUM_y	Equation (11)	0	342	–
Encoder	Figure 6	$8000\mu s$	1031	–
Min. Loop rate	Figure 19	$12000\mu s$	–	–

in Table 3, in which the largest consumed time is $8[ms]$ which is required for encoding the feedback signal. As mentioned in Section 4.1, the work of the paper relied on avoiding the use of look-up tables due to their relative difficulty, but also the comparison in the execution time calculations revealed a significant increase of up to $12[ms]$, i.e. over the encoding time that negatively affect the control of fast systems, see Table 4. The impact of the FPGA loop rate on the performance of the FLC's VIs is illustrated in Figure 25 with the loop rates of $\delta t = 10, 20, 50[ms]$. The results indicate that faster loop rates lead to an improved performance. The utilization of encoders featuring high resolutions has the potential to reduce the duration of converting the pulses per second into *rad/s*, thereby leading to an enhancement in performance. Tables 3 and 4 show the number of FPGA resources that utilized to construct the FLC-FPGA controller using the proposed VIs and using the look-up tables.

5.2. Two degrees of freedom controller

The autonomy of the two DaNI motors, see Figure (26), allows not only driving but also manoeuvrability of the robot. Thus the controller for each motor should be

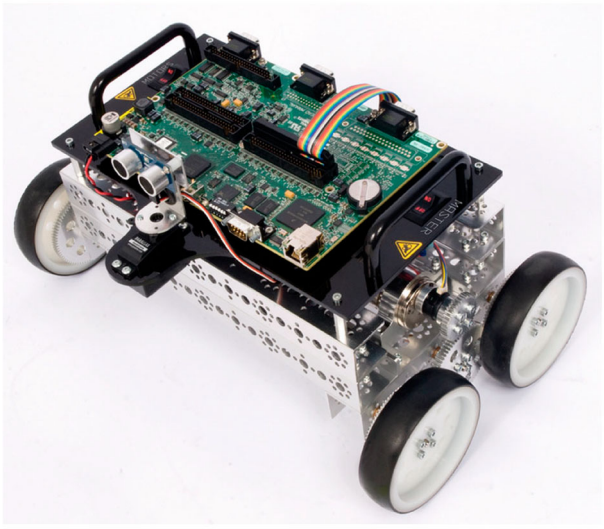

Figure 26. DaNi mobile robot.

independent, and the control signal should be calculated for each motor separately. Here, the effectiveness of the LabVIEW in programming and communicating with the hardware targets becomes clear; the control action can be calculated by adding one more parallel loop or through the For loop structure.

Concerning the trajectory shown in Figure (27), half-circular trajectories are usually imposed in obstacle avoidance algorithms. The angle $\psi(t)$ can be estimated such that $\psi(t_0) = 0$, $\psi(t_f) = \pi$, $\dot{\psi}(t_0) = \dot{\psi}(t_f) = 0$, $\ddot{\psi}(t_0) = \ddot{\psi}(t_f) = 0$. The obtained values of ψ are used to calculate the position and orientation of the robot in the specified path. The orientation of the robot is selected to be in the tangent direction of the radial direction, the constraints function can be stated as follows [40]:

$$\mathbf{C}^D(\mathbf{q}, \mathbf{t}) = \begin{bmatrix} R_x - a + R\cos\psi(t) \\ R_y - b + R\sin\psi(t) \\ \theta - \psi(t) - \frac{\pi}{2} \end{bmatrix} = \mathbf{0} \qquad (23)$$

Defining the wheel radius as r, and the wheel has a distance ℓ from the origin point. Given r, ℓ, θ, the velocity kinematics models can be constructed to predict the robot's overall speed in the global reference frame. The generalized coordinates vector is defined as $\mathbf{q} = \begin{bmatrix} R_x & R_y & \theta \end{bmatrix}^T$ and the operational velocity vector takes the form of $\dot{\mathbf{q}} = \begin{bmatrix} V_x & V_y & \omega \end{bmatrix}^T$ that can be evaluated based on the constraints of Equation (23). The velocity transformation between the motors angular velocities and the generalized velocities of the robot carrier can be expressed as

$$\begin{bmatrix} \dot{\phi}_R \\ \dot{\phi}_L \end{bmatrix} = \begin{bmatrix} 2/r\cos\theta & 2/r\cos\theta \\ 2/r\sin\theta & 2/r\sin\theta \\ 2\ell/r & -2\ell/r \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} \qquad (24)$$

Given the generalized velocity vector, $\dot{\mathbf{q}}$, the tracking problem's reference values for the robot's controller for the right and left wheels, $\dot{\phi}_R$ and $\dot{\phi}_L$, are estimated by using Equation (24). The following results consider the implementation of the preliminary design of FLC-FPGA and the PID-FPGA controllers, that discussed earlier in Figure 22, upon the motion of the robot for tracking the trajectory of Equation (23), and with the loop rate of 10[ms]. Figure 28 shows the results of the assigned controllers upon the robot's spinning velocities $\dot{\phi}_R$ and $\dot{\phi}_L$. Figures 29 and 30 show the results of the assigned controllers upon the generalized velocities. In the case of implementing PID-FPGA, it should be noted that the two motors are controlled by the identical set of PID control parameters, providing that both motors have the same dynamic model. However, the two motors' dynamics actually differ. As a result, the regulated controlled results clearly distinguish between the right and left motors. The right motor in the PID-FPGA in Figure (28) oscillates more in relation to the reference value. With the PID-FLC-FPGA, because the controller design is independent of the system model, the results are more accurate, especially during starts and stops. As illustrated in Figure (27), the robot's orientation has to be initiated at $\pi/2$ degrees and terminate at $3\pi/2$, as they were shown in Figure 31, of the real-time orientation of the robot, that demonstrates an error margin of the robot orientation below 0.5 degrees for both controllers. It is worth mentioning that the implementation of a control system for regulating the position of the robot was not carried out. The control entirely takes place as a velocity control upon the robot's motors, and the robot's position with respect to the intended trajectory serves as a measure of the effectiveness of the control systems that are implemented on these motors as subsystems. Based on this, the percentage of error is clear when drawing the Cartesian coordinates of the aforementioned control systems (PID-FPGA and FLC-FPGA-1). In the case of using the PID-FPGA, the error in the x-direction reaches to $-2[mm]$ and $+16[mm]$ in the y-direction , while the FLC-FPGA-1 controller exhibits a comparatively lower error of in x and y directions as $-1[mm]$ and $-12[mm]$ respectively, see Figures 32 and 33.

The effectiveness of the proposed FLC-FPGA controller in the presence of external disturbance is the true assessment, and therefore the subsequent investigation discusses this issue. The fuzzy controller with maximized rules and scaling constants (FLC-FPGA-3) is implemented upon the robot system for tracking the circular trajectory (Equation 23), and an external disturbances are introduced at approximately 15 and 21 seconds. The effect of implementing the (FLC-FPGA-3) upon the DC-motor is presented in Figure 24 that shows fastest response, smallest overshoot and minimum amount of error among other controllers. Specifically, an overshoot of 2% and settling time of 0.85[s]. Figure 34 demonstrates a high level of functionality, as

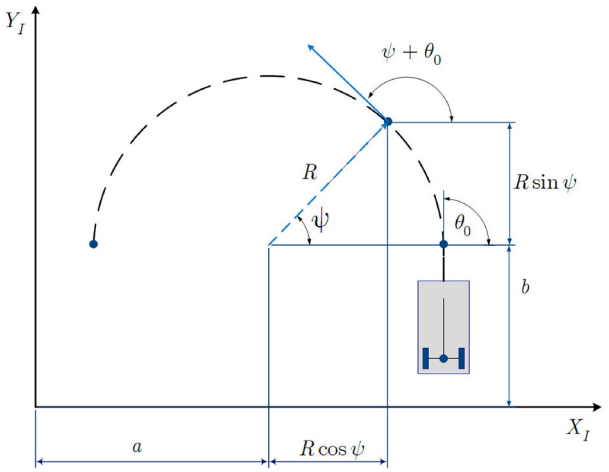

Figure 27. Desired circular trajectory.

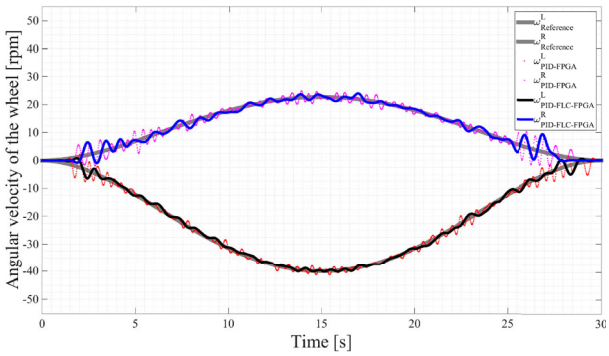

Figure 28. Angular velocity of the robot's wheels.

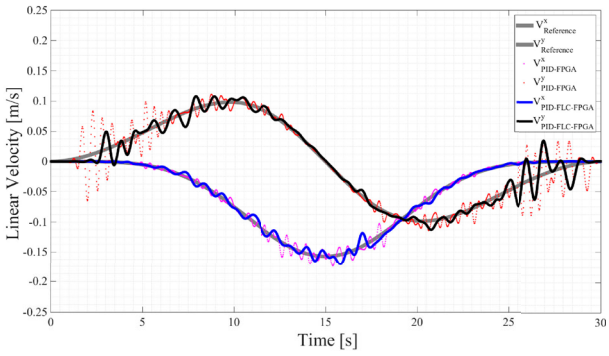

Figure 29. Robot's linear velocity.

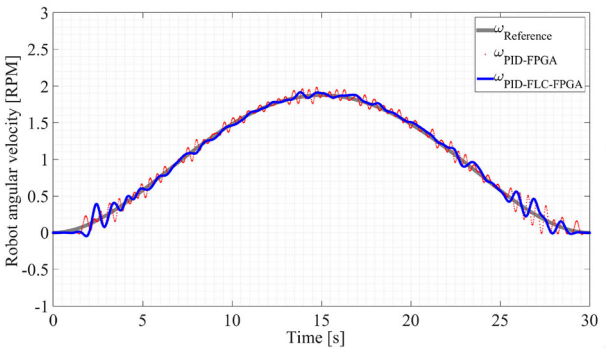

Figure 30. Robot's angular velocity.

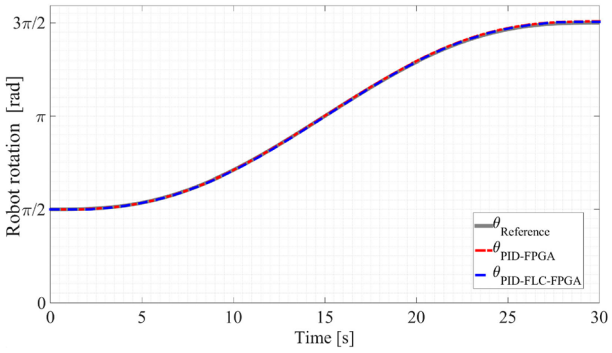

Figure 31. Robot's angular displacement.

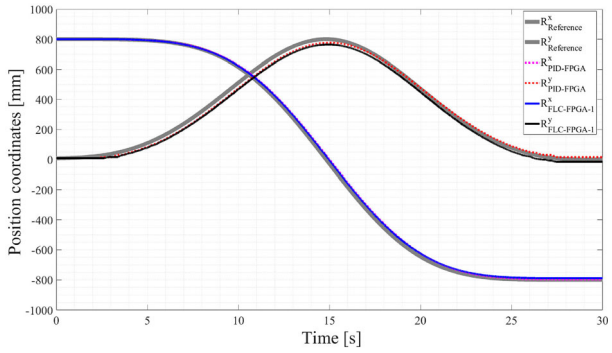

Figure 32. Robot's Cartesian position.

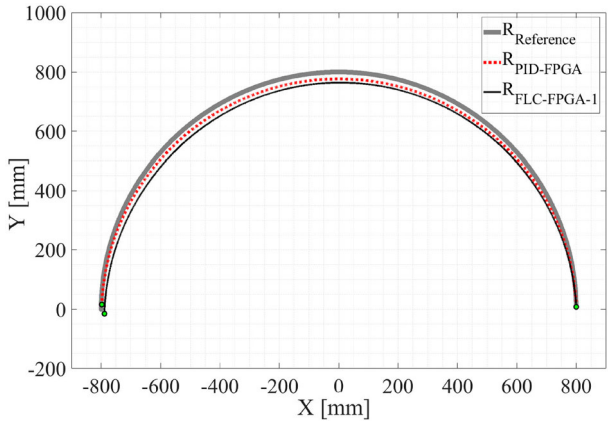

Figure 33. Trajectory tracking of the robot.

evidenced by the rapidity with which wheel responses are observed to compensate to disturbances. The effect of external disturbances appears across Figures 34–37. It is noteworthy that, at a time of 15 seconds, the instance of changing the direction of the robot, see Figure 37, the effect of this disturbance, appeared in the V^y component and did not appear horizontal direction V^x. In the case of the second 21, this effect appeared in both velocity components when the robot was subjected to the external disturbance. The robot positioning figures, i.e. Figures 36–37 show distinctive characteristics compared to PID-FPGA and FLC-FPGA-1, whereby the error reaches zero at the end point of the trajectory, not in normal operation but in the presence of disturbances.

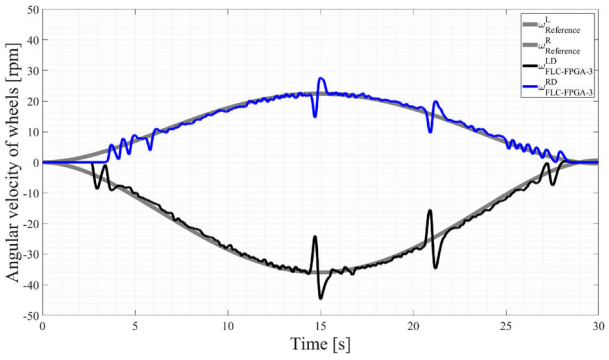

Figure 34. Effect of disturbance on the robot's wheels.

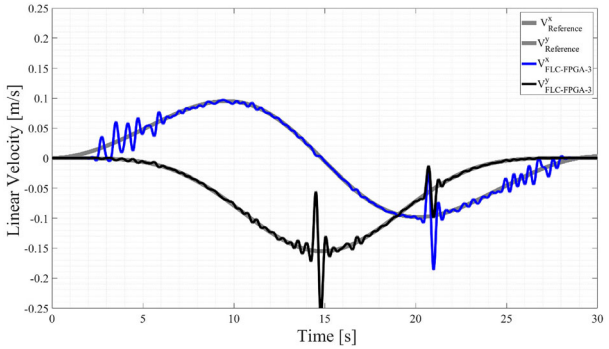

Figure 35. Effect of disturbance on the linear velocity.

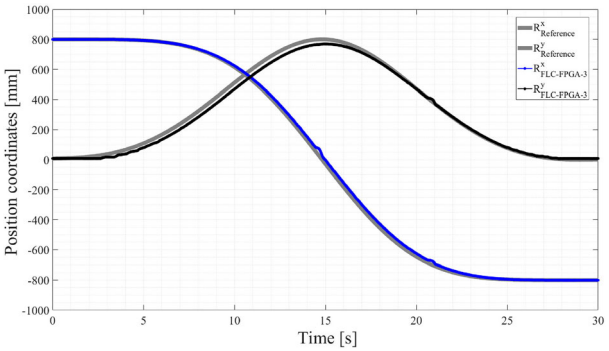

Figure 36. Effect of disturbance on the linear position.

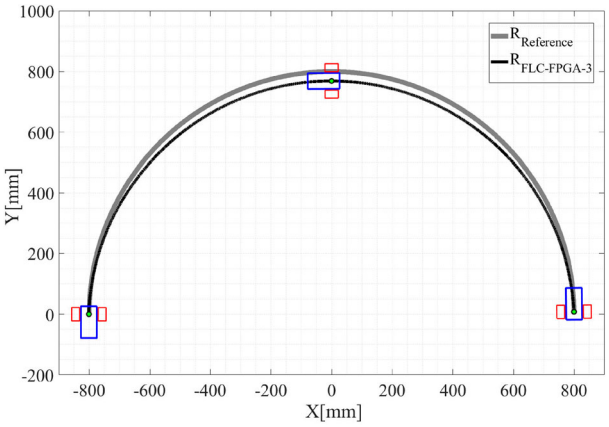

Figure 37. Effect of disturbance on the robot's track.

6. Conclusion

The Fuzzy Logic Control (FLC) is implemented in the current study on an FPGA target, which has several benefits for high-speed embedded applications. The computation of the control law may be challenging when a standalone controller is implemented, as opposed to Hardware-in-the-Loop (HIL) applications. LabVIEW FPGA does not support floating point data types. It only uses fixed math operations on integer data types. Complex mathematics is difficult to implement on FPGA due to the target device's lack of floating point operations. This paper provides a brief introduction to deploying FLC methods using Reconfigurable Inputs/Outputs (RIO) FPGA technology. It proposes a method of implementing the three stages that constituent the FLC and the PD-FLC and PID-FLC structures as well. Both 1- and 2-degrees-of-freedom controllers are developed and implemented upon step reference, controlling the speed of the DC-motor system and tracking a differential-wheeled mobile robot. The experimental results and performances of the PD-FLC-FPGA and PID-FLC-FPGA are compared with the classical PID-FPGA controller for reference speed and tracking problems. The preliminary selection of rules is defined, suitable for motion control of mechatronic systems. Furthermore, the stability criteria are established to enable the user to choose the initial scaling coefficients. Moreover, the rules and scaling factors of the PID-FLC-FPGA controller can be optimized using a genetic algorithm without affecting the embedded controller structure. The optimized PID-FLC-FPGA controller exhibits an overshoot of 2% and settling time of $0.85[s]$ against 32% and settling time of $1.5[s]$ for the PID-FPGA controller. The proposed controller, in its optimized version, shows minimum error, and consequently reduce the energy demanded. In addition, it produce an excellent attenuation of disturbance rejection, resulting in the elimination of trajectory tracking errors. In conclusion, the experimental results validate the control design procedure and implementation and additionally show an improvement of the PID-FLC-FPGA over the PID-FPGA in both tests. This study could be useful in the effective integration of intelligent control systems with FPGA technology to create embedded systems for fast mechatronics applications.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Note

1. www.xilinx.com.

ORCID

Ayman A. Nada http://orcid.org/0000-0002-1873-5756

References

[1] Hou Z-S, Wang Z. From model-based control to data-driven control: survey, classification and perspective. Inf Sci. 2013;235:3–35. doi: 10.1016/j.ins.2012.07.014

[2] Wang L. A course in fuzzy systems and control. Englewood Cliffs, New Jersey: Prentice Hall; 1997.

[3] Somwanshi D, Bundele M, Kumar G, et al. Comparison of fuzzy-pid and pid controller for speed control of dc motor using labview. Procedia Comput Sci. 2019;152:252–260. doi: 10.1016/j.procs.2019.05.019

[4] Chantrapornchai C, Pipatpaisan J. Fuzzy application parallelization using openmp, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 6132 LNCS; 2010, p. 122–132.

[5] Qasaimeh M, Sagahyroon A, Shanableh T. A parallel hardware architecture for scale invariant feature transform (sift), International Conference on Multimedia Computing and Systems -Proceedings. 2014. p. 295–300.

[6] Li J, Pan Y. Gpu-based parallel optimization for real-time scale-invariant feature transform in binocular visual registration. Pers Ubiquitous Comput. 2019;23:465–474. doi: 10.1007/s00779-019-01222-3

[7] Park H, Kim S. Hardware accelerator systems for artificial intelligence and machine learning. Adv Comput. 2021;122:51–95. doi: 10.1016/bs.adcom.2020.11.005

[8] Song WJ. Hardware accelerator systems for embedded systems. Adv Comput. 2021;122:23–49. doi: 10.1016/bs.adcom.2020.11.004

[9] Dubey R. Introduction to embedded system design using field programmable gate arrays. London: Springer; 2009.

[10] Rosero JR, Gonzalez GR, Khanna R. Field programmable gate array applications – a scientometric review. Computation. 2019;7:63. doi: 10.3390/computation7040063

[11] Miguel EEC, Resendiz JR, Martinez JRG, et al. Field – programmable gate array – based laboratory oriented to control theory courses. Comput Appl Eng Edu. 2019;27:1253–1266. doi: 10.1002/cae.v27.5

[12] Ortiz A, Mendez E, Balderas D, et al. Hardware implementation of metaheuristics through labview fpga. Appl Soft Comput. 2021;113:107908. doi: 10.1016/j.asoc.2021.107908

[13] Kandidayeni M, Macias A, Trovao JP, et al. Control systems with compact-rio implementation. In: Reference Module in Materials Science and Materials Engineering. Elsevier; 2022.

[14] Hemalatha B, Ravi VR, Divya S, et al. Embedded fpga controller for robot arm in material handling using reconfigurable nicompact rio. IEEE; 2014. p. 125–131.

[15] dos Santos MPS, Ferreira J. Novel intelligent real-time position tracking system using fpga and fuzzy logic. ISA Trans. 2014;53:402–414. doi: 10.1016/j.isatra.2013.09.003

[16] Antonio-Mendez R, de la Cruz-Alejo J, Penaloza-Mejia O. Fuzzy logic control on fpga for solar tracking system. Mech Mach Sci. 2015;25:11–21. doi: 10.1007/978-3-319-09858-6

[17] Akbat O, Uzgun HD, Akkaya S. Hardware-in-the-loop simulation and implementation of a fuzzy logic controller with fpga: case study of a magnetic levitation system. Trans Inst Meas Control. 2019;41:2150–2159. doi: 10.1177/0142331218813425

[18] Bourhnane S, Abid MR, Lghoul R, et al. Real-time control of smart grids using ni compactrio. IEEE; 2019. p. 1–6.

[19] Contreras-Medina L, Romero-Troncoso R, Millan-Almaraz J, et al. Fpga based multiple-channel vibration analyzer embedded system for industrial applications in automatic failure detection. IEEE; 2008, p. 229–232.

[20] Nada AA, Shaban EM. The development of proportio nal-integral-plus control using field programmable gate array technology applied to mechatronics system. Am J Res Commun. 2014;2:14.

[21] Shaban EM, Nada AA. Proportional integral derivative versus proportional integral plus control applied to mobile robotic system. J Am Sci. 2013;9:583–591.

[22] Ponce-Cruz P, Molina A, MacCleery B. Fuzzy logic type 1 and type 2 based on LabVIEW FPGA. Cham, Switzerland: Springer International Publishing; 2016.

[23] Ponce-Cruz P, Molina A, MacCleery B. Fuzzy logic type 1 and type 2 labview fpga toolkit. Stud Fuzziness Soft Comput. 2015;334:159–230. doi: 10.1007/978-3-319-26656-5

[24] Garrido R, Calderon D, Soria A. Adaptive fuzzy control of dc motors, International Power Electronics Congress – CIEP; 2006. p. 158–163.

[25] Sanjay D, Kumar PR, Savithri TS. Fuzzy control for person follower fpga based robotic system. Indian J Sci Technol. 2015;8(23):1–5. doi:10.17485/ijst/2015/v8i23/85333.

[26] Singh DA, Singh RK, Sharma S. Analogous research of classical pi controller and fuzzy logic controller to control the speed of d.c. servo motor. J Phys Conf Ser. 2021;2007:012044. doi: 10.1088/1742-6596/1714/1/012044

[27] Harshitha S, Shamanth S, Chari AK. A review of various controller techniques designed for the operational control of dc and servo motors. J Phys Conf Ser. 2022;2273:012001. doi: 10.1088/1742-6596/2273/1/012001

[28] H. M. Ramadan EAE, El-Bardini M, El-Rabaie NM, et al. Embedded system based on a real time fuzzy motor speed controller. Ain Shams Eng J. 2014;5:399–409. doi: 10.1016/j.asej.2013.10.001

[29] García-Montalva JC, de la Cruz-Alejo J, Díaz-Salgado J. Fuzzy logic control on fpga using labview. Mech Mach Sci. 2015;25:261–271. doi: 10.1007/978-3-319-09858-6

[30] Siddique N, Adeli H. Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing. 1st ed. Hoboken, New Jersey: John Wiley and Sons, Ltd; 2013.

[31] Ziouzios D, Kilintzis P, Baras N, et al. A survey of fpga robotics applications in the period 2010–2019. Adv Sci Technol Eng Syst J. 2021;6:385–408. doi: 10.25046/astesj

[32] National instruments – getting started with the fpga module, 2023. https://www.ni.com/pdf/manuals/374737k.html, last visited 2023-03-16.

[33] Wang G, Andrade HA. Labview: a graphical system design environment for adaptive hardware/software systems. IEEE; 2010, p. 82–82.

[34] Perry AG, Feng G, Liu YF, et al. A design method for pi-like fuzzy logic controllers for dc-dc converter. IEEE Trans Indust Electron. 2007;54:2688–2696. doi: 10.1109/TIE.2007.899858

[35] Pati S, Patnaik M, Panda A. Comparative performance analysis of fuzzy pi, pd and pid controllers used in a scalar controlled induction motor drive, 2014 International Conference on Circuits, Power and Computing Technologies, ICCPCT; 2014, 2014. p. 910–915.

[36] Calderon CA, Sarango R, Macas E, et al. Implementation and comparative analysis of fractional order pid embedded controllers, applied to speed control of a robotic prosthesis. IEEE; 2018. p. 1–6.
[37] Wong SV, Hamouda AM. Optimization of fuzzy rules design using genetic algorithm. Adv Eng Softw. 2000;31:251–262. doi: 10.1016/S0965-9978(99)00054-X
[38] Sio KC, Lee CK. Stability of fuzzy pid controllers. IEEE Trans Syst Man Cybern A Syst Hum. 1998;28:490–495. doi: 10.1109/3468.686710
[39] Carvajal J, Chen G, Ogmen H. Fuzzy pid controller: design, performance evaluation, and stability analysis. Inf Sci. 2000;123:249–270. doi: 10.1016/S0020-0255(99)00127-9
[40] Nada AA, Bashiri AH. Integration of multibody system dynamics with sliding mode control using fpga technique for trajectory tracking problems. Georgia, USA: American Society of Mechanical Engineers; 2018. https://doi.org/10.1115/DSCC2018-9108