University of Alberta

Design and FPGA Implementation of a Log-Domain High-Speed Fuzzy Control System

by

Md Ali Razib

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Electrical and Computer Engineering

Spring 2010
Edmonton, Alberta

Examining Committee

Scott Dick, Electrical and Computer Engineering

Vincent Gaudet, Electrical and Computer Engineering

Marek Reformat, Electrical and Computer Engineering

Nelson Amaral, Computing Science

Abstract

The speed of fuzzy controllers implemented on dedicated hardware is adequate for control of any physical process, but too slow for today's high-complexity data networks. Defuzzification has been the bottleneck for fast implementations due to the large number of computationally expensive multiplication and division operations. In this thesis, we propose a high-speed fuzzy inferential system based on log-domain arithmetic, which only requires addition and subtraction operations. The system is implemented on a Xilinx Virtex-II FPGA with a processing speed of 67.6 MFLIPS having a maximum combinational path delay of 4.2 ns. It is a clear speedup compared to the reported fastest 50 MFLIPS implementation. A pipelined version of the controller is also implemented, which achieves a speed of 248.7 MFLIPS. Although a small approximation error is introduced, software simulation and hardware implementation on FPGA confirm high similarity of the outputs for control surfaces and a number of second-order plants.

Acknowledgments

I express my sincere gratitude to Russell Dodd, Eric Son and Ji Sun for their valuable suggestions during the implementation of log-domain controller on FPGA.

Dedication

In loving memory of my father

Md Farman Ali

(1944 - 2007)

TABLE OF CONTENTS

List of Tables

List of Figures

List of Symbols/ Acronyms

DSP:	Digital Signal Processing
FLIPS:	Fuzzy Logic Inferences Per Second
FPGA:	Field Programmable Gate Array
VLSI:	Very Large Scale Integration
PI:	Proportional-Integral
PD:	Proportional-Derivative
PID:	Proportional-Integral-Derivative
LUT:	Look-Up Table
COG:	Center Of Gravity
MOM:	Mean Of Maxima
PWL:	Piece Wise Linear
MOS:	Metal-Oxide Semiconductor
CMOS:	Complementary Metal Oxide Semiconductor
NLL:	Normalized Locked Loop
I-V:	Current to Voltage

FPAA:	Field Programmable Analog Array
A/D:	Analog to Digital
D/A:	Digital to Analog
VLSI:	Very Large Scale Integration
ROM:	Read-Only Memory
RAM:	Random Access Memory
ASIC:	Application-Specific Integrated Circuit
FPE:	Fuzzy Processing Element
FALU:	Fuzzy Arithmetic Logic Unit
KAFA:	KAist Fuzzy Accelerator
FRHC:	Fired Rules Hyper Cube
DA:	Distributed Arithmetic
DC:	Direct Current

CHAPTER 1
INTRODUCTION

Data communication networks, and particularly the Internet, play a pivotal role in modern society. To improve effective bandwidth, Quality of Service (QoS) and other performance metrics, the optimization of these networks is very important. A number of network optimization goals can be represented as control problems, and solved by traditional control algorithms. However, one drawback of traditional control is that it requires an exact mathematical model of a system, and this model is often unavailable for large and complex networks. Intelligent control systems that do *not* require exact models have become more commonplace. Fuzzy logic controllers, proposed in theory by Zadeh [9], [10], and first implemented in automatic control by Mamdani [11], [12], are a particular class of intelligent controllers built on fuzzy expert systems, that show substantial performance improvement in simulation over standard algorithms for these problems.

The implementation of a fuzzy controller, however, is not straightforward. Software simulations on general-purpose microprocessors are typically slow due to the lack of instruction-set support for fuzzy operations. This imposes a big problem for today's high-complexity and high-speed networks, requiring specialized hardware. A number of researchers have pursued fuzzy hardware development, beginning with a voltage-mode analog circuit in 1969. Although analog implementations [51] provide lower power consumption and lower chip size [50], they suffer from being slower, less accurate, less flexible and less scalable compared to their digital counterparts. Thus research into digital hardware implementations is ongoing, with reported performance of ASIC implementations ranging from 20 to 50 million fuzzy logic inferences per second (MFLIPS) [25]. This is a commonly used performance measure for fuzzy

systems[1]. It indicates the rate at which a defuzzified output can be computed [24]. Digital fuzzy hardware implementations, however, still have a bottleneck due to the large number of costly multiplication and division operations required in the defuzzifier, the last module in the fuzzy controller [22].

In this thesis, we propose and implement a high-speed fuzzy controller on hardware based on log-domain arithmetic. The basic principle of our approach is to represent all quantities by their logarithmic values, therefore transforming multiplication and division functions into computationally much simpler addition and subtraction operations. The logarithmic values are stored in lookup tables to facilitate instant conversion between a log value and its exponential value. The summation of a number of log-domain values is approximated as the maximum logarithmic value to simplify calculations. Although it introduces a small approximation error, simulation and hardware implementation results illustrate the similarity of the generated outputs with the expected outputs. Our system outperforms the reported fastest implementation in the literature [80] by 33% in speed. An even higher processing speed is achieved for a pipelined version, which we postulate would enable our system to be applicable in high-speed data networking applications.

The rest of the thesis is organized as follows. Chapter 2 starts with an overview of control systems and then goes on to discuss the concepts behind fuzzy controllers and their analog and digital hardware implementation. A review of published literature on fast defuzzification is also presented. Chapter 3 discusses the design of our log-domain controller and the justification of using a correction factor to improve the results. Simulation results for a number of second-order plants are detailed in Chapter 4. It also includes performance comparisons with a linear PD and a typical fuzzy controller. Chapter 5 presents the hardware implementation with speed and resource usage comparisons between a pipelined and a non-

[1] MFLIPS are difficult to use to make comparisons since the reported number is highly dependent on the number of rules in the rule base and the computational precision.

pipelined version of the log-domain controller. Chapter 6 provides directions for future research, and Chapter 7 concludes the thesis.

CHAPTER 2
BACKGROUND LITERATURE

2.1 Overview of Control Systems

A control system is an interconnection of components forming a system configuration to generate a desired system response. Control systems deal with the behavior of dynamic systems – systems for which outputs change with time, and are a factor of the inputs. In a control system, the desired output is called the *reference* [1]. A control system produces an output for a given input, where the input and output represent the desired response and the actual response respectively. There are numerous examples of control systems being used, such as home heating and air-conditioning systems controlled by a thermostat, the cruise control of an automobile, an elevator-position control system used in high-rise multi-level buildings, human heart using a pacemaker and so on [42]. The device or process to be controlled is termed as a *plant*. The plant, for instance, can be a furnace system where temperature is the output variable. Usually the plant output is sent back for comparison with the reference to produce the next input, as shown in Figure 2.1.

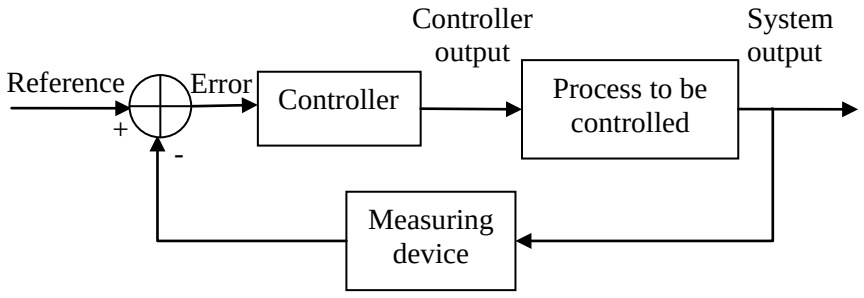


Figure 2.1: Basic control system

2.1.1 Basic Feedback Control

Control systems are typically classified as open-loop or closed-loop systems. Open-loop control systems do not monitor or correct the output for disturbances, while closed-loop system keep monitoring the output and comparing it with the input. Figure 2.2 shows a general open-loop control system.

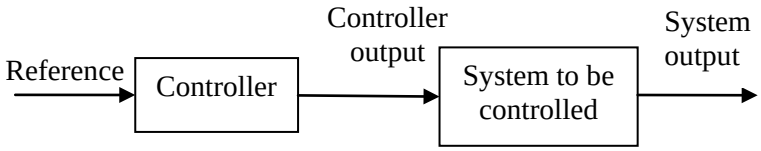


Figure 2.2: Open-loop control system

As seen from Figure 2.2, there is no way for the output to get fixed automatically if it is not at an acceptable value. Any parameter having a different value than what was anticipated causes a proportional error in the output. For instance, let us consider a gas furnace as a control system [2]. In case of no adequate temperature sensor, manually calibrating the valve is the only way to adjust the gas flow. This system would only work on average days when the room temperature remains steady. However, if external weather conditions change, the furnace would not be able to provide the desired temperature. An open-loop controller is usually an analytical tool used to design a real closed-loop system.

Closed-loop (feedback) control systems are similar to open-loop ones, except for the fact that a sensor monitors the output and it is compared with the input signal. The difference between the input and output signal is referred to as the error. An ideal feedback system produces the desired response exactly by cancelling out all errors. Continuing with the previous example, suppose a thermostat is there to measure the temperature and can be set at any required value. So for normal temperature, the system works as the open-loop system does; however, it provides greatest benefits when there is a change in temperature. The thermostat increases the valve opening if the temperature decreases, and vice versa. This dynamic

compensation for changes in the output is the principle behind the feedback control system. A closed-loop control system [38] is shown in Figure 2.3.

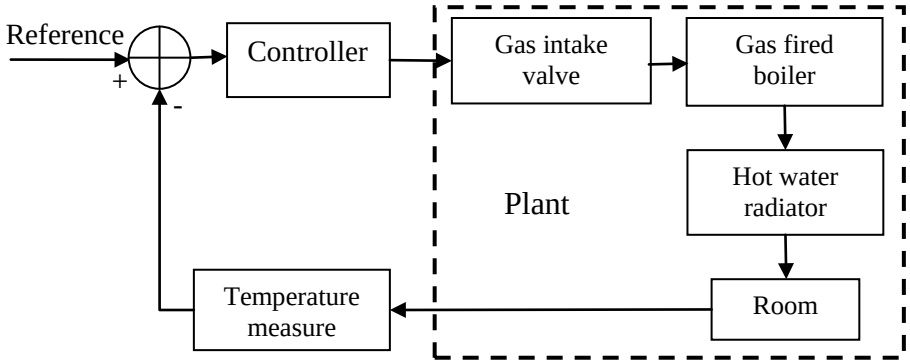


Figure 2.3: An example of feedback control system [38]

2.1.2 Modes of Control

Modes of control are the methods to calculate the values of the controller output variables. The modes are: proportional (P), integral (I) and derivative (D). They are most commonly used in some combination, although single modes are also possible. Among the different combinations, proportional-integral-derivative (PID) controller is very widely used in industrial control systems. Figure 2.4 shows the block diagram of a PID controller. Note that there are other types of traditional control systems available, namely – phase-lead, phase-lag and quadratic optimal controller, besides the commonly used PID controllers.

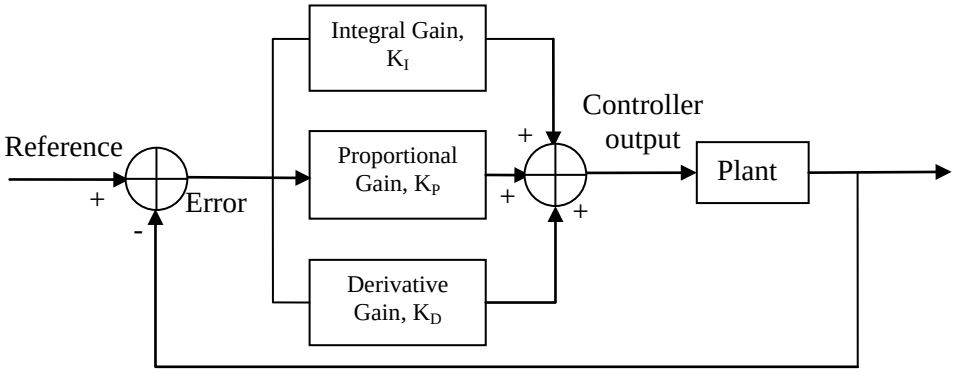


Figure 2.4: PID Controller

The proportional part calculates the reaction to the current error, the integral part considers the sum of recent errors to determine the reaction, and the derivative part focuses on the rate at which the error has been changing. The gains, K_P, K_I and K_D, represent the weights applied to each part respectively before summing them up. To achieve specific design requirements the constant gains are tuned. In some cases, only one or two modes are needed to provide the appropriate output. This is done simply by settling the undesired gain to zero. The derivative gain is often disabled to obtain PI controllers [3]. The individual modes are discussed below.

2.1.2.1 Proportional Mode

When a controller is in proportional mode, the controller output is, as the name suggests, proportional to the measured value. It does not keep track of the history of the value measured neither does it account for the rate of change in measured value. Tuning the purely proportional controller for the desired performance is relatively easy; however, a steady-state error exists between the set point and measured value in most cases. A proportional (P) controller is shown in Figure 2.5.

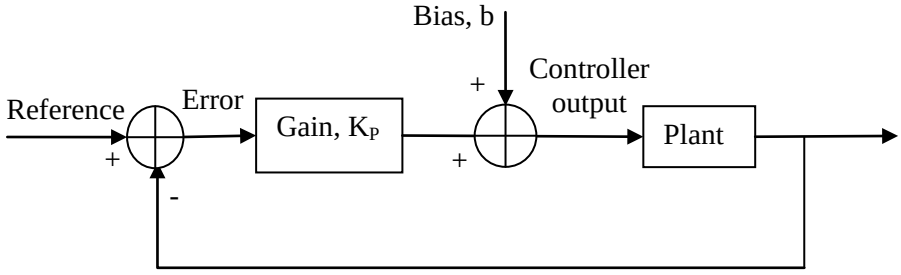


Figure 2.5: Proportional controller [3]

The behavior is expressed by the following equation:

$$U(t) = k_P\, e(t) + b \tag{2.1}$$

where $U(t)$ is the controller output, k_p is the proportional gain, $e(t)$ is the error and b is the output bias.

Bias or manual reset provides the value of the output when the error is zero.

2.1.2.2 Integral Mode

The contribution from the integral term is proportional to both the magnitude and duration of the error. An integral (I) mode controller consists of an integrator. Thus if the error signal (input to the integrator) is positive, the integrator output shifts upward, and it shifts downward for negative input. The output remains steady when measured value is equal to the set point. In comparison with the proportional controller, an integral controller does not need manual reset to adjust the output bias. Now the bias is automatically set by the output of the integrator. Figure 2.6 shows a PI controller.

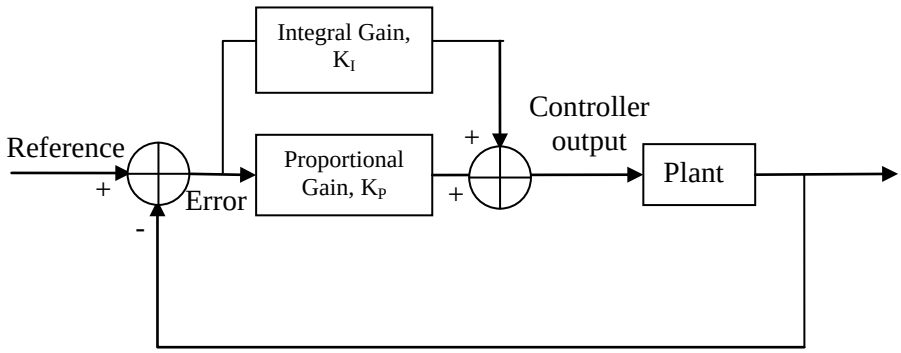


Figure 2.6: PI controller

The equation for PI controller is:

$$U(t) = k_P\, e(t) + K_I \int e(t)dt \qquad (2.2)$$

where $U(t)$ is the controller output, k_p is the proportional gain, K_I is the integral gain and $e(t)$ is the error.

The integral term (added with the proportional term) accelerates the movement of the process towards the set point and eliminates the residual steady-state error. However, the integral term can cause overshooting (cross over the set point and create a deviation in the other direction).

2.1.2.3 Derivative Mode

Derivative term calculates the rate of change of error and multiplies it by the gain K_D. The derivative term reduces the speed of the rate of change of the controller output, hence is used to reduce the magnitude of overshoot generated by the integral term.

The equation for a PID controller is:

$$U(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \tag{2.3}$$

A tuning parameter, T_D can be used to adjust the relative effect of the derivative mode. However, derivative term can make the system unstable if the noise and derivative gain are sufficiently large [3].

2.1.3 Laplace Transformation

A differential equation is one way to describe, in mathematical terms, the relationship between the input and output of dynamic systems. It depends on the components which form the dynamic system and the manner in which they are connected [2]. It is a relationship between a function of time and its derivatives. For instance, the equations $\frac{dy}{dt} = 3y^2 \sin(t+y)$ and $\frac{d^3y}{dt^3} = e^{-y} + t + \frac{d^2y}{dt^2}$ are two examples of differential equations [41]. The order of a differential equation is the order of the highest derivative of the function *y* appearing in the equation. Therefore the previous equations are first and third order differential equations, respectively.

A classic example of developing a differential equation is provided in [2] with a freely falling body through the air towards the Earth. There are two active forces being applied to the object. One is the weight of the object, $W = mg$, where *m* is the mass of the object and *g* is the acceleration of gravity. The other force is the air resistance acting on the object opposing the motion. To simplify things, a linear relationship is assumed between the friction force and speed, where the

friction force is equal to the product of velocity and the friction coefficient *b*, that is,

Friction $= b\frac{dy}{dt}$, where *y* is the position of the object at any value of time *t*.

Now, applying Newton's second law which states that force is equal to mass times acceleration,

$$\sum forces = m\frac{d^2y}{dt^2} \tag{2.4}$$

where $\frac{d^2y}{dt^2}$ is the acceleration.

Since friction force is opposite to the weight, from (2.4),

$$W - Friction = m\frac{d^2y}{dt^2}$$

Or, $m\frac{d^2y}{dt^2} + b\frac{dy}{dt} = mg$ (2.5)

This is the resulting differential equation. A differential equation can be either linear or nonlinear. A linear differential equation is any equation that can be written in the form [39]:

$$a_n(x)\frac{d^ny}{dx^n} + a_{n-1}(x)\frac{d^{n-1}y}{dx^{n-1}} + ... + a_1(x)\frac{dy}{dx} + a_0(x)y = F(x) \tag{2.6}$$

where $a_n(x)$, $a_{n-1}(x)$, …, $a_1(x)$, $a_0(x)$ and *F(x)* depend only on the independent variable *x*, not on *y*. For example, $\frac{d^2y}{dx^2} + y = x^3$ is a linear second-order differential equation, while $\frac{d^2y}{dx^2} + \cos y = 0$ is nonlinear.

A nonlinear differential equation does not have a solution in most cases; therefore, in practice, the equation is approximated as a linear one [39]. That is why linear differential equations are used to represent practical controllers. As the order of the differential equations grows, it becomes increasingly difficult and almost impossible to solve them using standard integration techniques. First order equations, the simplest ones, are categorized into several classes to be applicable to specific techniques of integration. For instance, if the right-hand side of the equation *dx = f(x, y)* can be expressed as a function that depends only on *x* times a function that depends only on *y*, then the equation is called separable equation and solved by standard integration methods [39]. For solving higher-order equations, some approximation methods such as Euler's method, Taylor and Runge-Kutta method, have been developed.

The difficulty in solving differential equations using standard techniques gave rise to an approach called Laplace transformation. It is a very attractive method, because it converts the solution's process into a series of algebraic operations. It is also ideal for solving linear differential equations with constant coefficients. A system described by such equations is termed a linear-time-invariant (LTI) system.

The Laplace transformation of a function of time *f(t)* is given by [5]:

$$L[f(t)] = \int_0^\infty f(t)e^{-st}dt = F(s) \tag{2.7}$$

Where $L[f(t)]$ is the shorthand notation for the Laplace integral and the parameter *s* is a complex quantity of the form $\sigma + j\omega$.

The transform takes a function of *t* and produces a function of *s*, which is done by multiplying *f(t)* by e^{-st} and then integrating with respect to *t* from 0 to ∞. However, *f(t)* must meet a couple of requirements to be Laplace-transformable. The requirements are – the function must be piecewise continuous over every finite interval $0 \le t_1 \le t \le t_2$, and the function must be of exponential order [5]. A

function is piecewise continuous in a finite interval if that interval can be divided into a finite number of subintervals and the function is continuous over each of the subintervals. *f(t)* must also possess finite limits at the ends of each subintervals. A function *f(t)* is of exponential order if there exists a constant *a* such that the product $e^{-at}|f(t)|$ is bounded for all values of *t* greater than some finite value *T*. This imposes the restriction that σ, the real part of *s*, must be greater than a lower bound σ_a for which the product $e^{-\sigma_a t} \mid f(t) \mid$ is of exponential order.

Once the differential equation in time domain is transformed into an equation in *s*-domain, it can be solved using algebraic manipulations. However, Laplace transformation is not an easy operation, except for very simple functions of *f(t)*. Fortunately, the results can be stored in a table after the operation is performed once. Finally, to complete the solution, the *F(s)* needs to be converted to an *f(t)* using inverse Laplace transform of *F(s)*. For this process, there is a corresponding operation, namely:

$$f(t) = L^{-1}[F(s)] = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} F(s)e^{st}ds \qquad (2.8)$$

As with the Laplace transform, lookup tables of inverse Laplace transforms for a number of common functions are published for convenience. When the response transform cannot be found in the tables, the general procedure is to express *F(s)* as the sum of partial fractions with constant coefficients. The partial fractions have a first-order or quadratic factor in the denominator and are readily found in the lookup tables. The complete inverse transform is the sum of the inverse transforms of each fraction.

2.1.4 Transfer Functions

Transfer functions are tools to describe the characteristics of dynamic components and systems. It provides a common language that allows engineers to communicate the behavioral aspects of a system. For a linear differential equation, the ratio of the output variable to the input variable is called the transfer function. If we take the Laplace transform of the high-order differential

equation $\frac{d^n y(t)}{dt^n} + a_{n-1}\frac{d^{n-1} y(t)}{dt^{n-1}} + ... + a_0 y(t) = b_{n-1}\frac{d^{n-1} u(t)}{dt^{n-1}} + ... + b_0 u(t)$, it becomes an algebraic equation of the form, with zero initial conditions [4]:

$$s^n Y(s) + a_{n-1}s^{n-1}Y(s) + ... + a_0 Y(s) = b_{n-1}s^{n-1}U(s) + ... + b_0 U(s) \quad (2.9)$$

If $A(s) = s^n + a_{n-1}s^{n-1} + ... + a_0$ and $B(s) = b_{n-1}s^{n-1} + ... + b_0$, we get *Y(s) = G(s)U(s)* where $G(s) = \frac{B(s)}{A(s)}$ and *G(s)* is the transfer function. The denominator polynomial *A(s)* is called *characteristic polynomial*. The roots of *A(s)* are the *poles* of *G(s)* and the roots of *B(s)* are the *zeros* of *G(s)*.

For instance, let us consider a differential equation of the form:

$$m\frac{d^2 y}{dt^2} + b\frac{dy}{dt} = w(t) \quad (2.10)$$

Taking the Laplace transform, assuming zero initial conditions,

$$ms^2 Y(s) + bsY(s) = W(s) \quad (2.11)$$

where *W(s)* is the Laplace transform of *w(t)*. Now, $\frac{Y(s)}{W(s)} = \frac{1}{ms^2 + bs}$ is the transfer function. Control systems and plants are usually represented by transfer functions.

2.1.5 Step Responses of Continuous-Time Linear Systems

The response of a control system to a sudden input is an important issue from a practical standpoint, because depending on the amount and rate of deviations from the long term steady state, they have varying degree of impacts on the overall system. In addition, the control system response is delayed until the output settles down to some vicinity of its final value. This makes the response to an impulse or Dirac Delta function an interesting topic in the study of linear systems. Dirac Delta is the limit as $\Delta \to 0$ of the pulse shown in Figure 2.7.

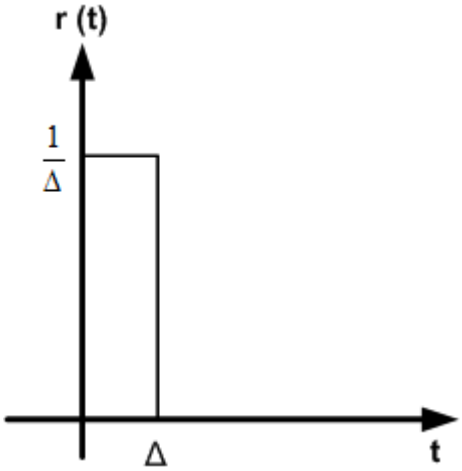


Figure 2.7: Discrete pulse

Since the Laplace transform of the Dirac Delta is 1, the transfer function of a continuous-time system is the Laplace transform of its response to an impulse with zero initial conditions. Because of the idealization implicit in the definition of an impulse, a system's dynamic behaviour is usually studied using the step response – the time behaviour of the outputs of a system when its inputs change from zero to one in a very short time.

For a sample step response shown in Figure 4.3 in [4], a set of parameters describing certain properties of the system is as follows:

Steady-state value is the final value of the step response.

Rise time is the time elapsed up to the instant at which the step response reaches, for the first time, 90% of the final value.

Overshoot, expressed as a percentage of the steady-state value, is the maximum instantaneous amount by which the step response exceeds its final value.

Undershoot is the absolute value of the maximum instantaneous amount by which the step response falls below zero. It is also expressed as a percentage of the steady-state value.

Settling time is the time elapsed until the step response enters a specific deviation band, $\pm\delta$, around the final value and does not get out of the band. The deviation is defined as percentage of the steady-state value.

A control system can respond to a step input in three ways – under damped, over damped and critically damped response.

Under Damped Response

The system will change the output quickly to the new value. However, there will always be an overshoot and the output will settle into the new value after a number of oscillations. Figure 2.8 shows an under damped response.

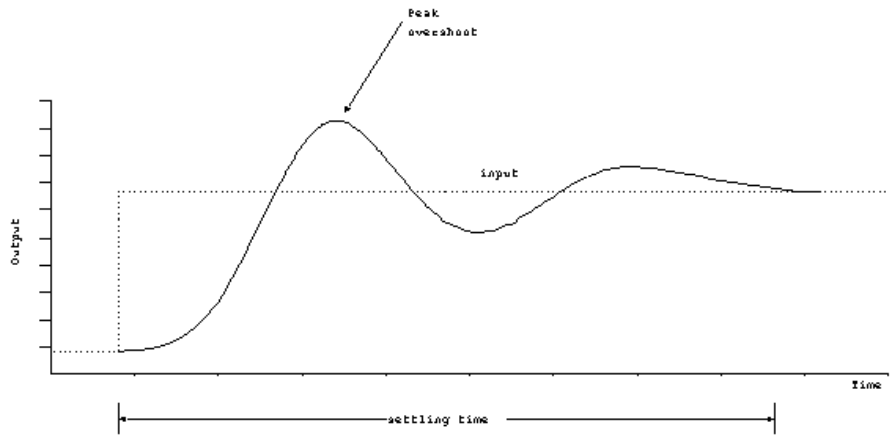


Figure 2.8: Under damped response [33]

Over Damped Response

In this case, there will be no overshoot, but the response is quite slow to reach the final value, illustrated by Figure 2.9.

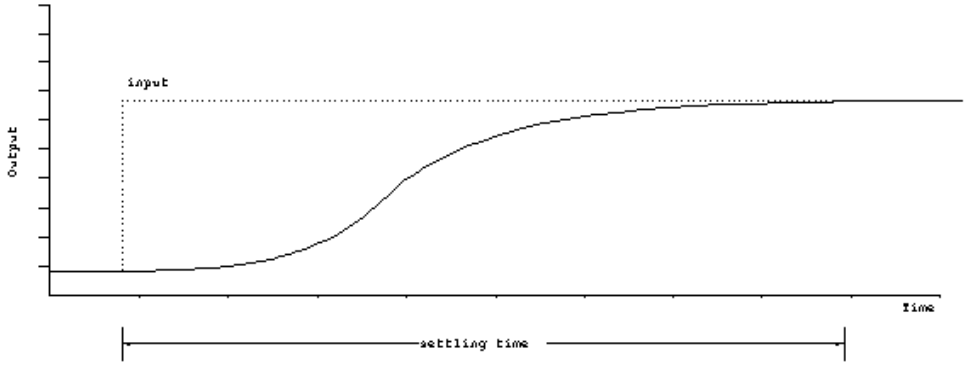


Figure 2.9: Over damped response [33]

Critically Damped Response

In this case, there may or may not be any overshoot, but there will be no oscillation. Critically damped system reaches the final value in the minimum amount of time. Figure 2.10 shows a critically damped response.

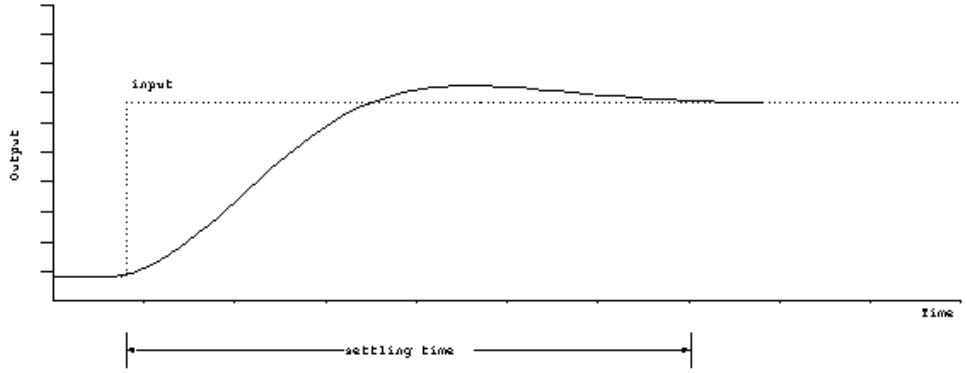


Figure 2.10: Critically damped response [33]

2.1.6 Digital Control Systems

Throughout the past few decades, modern control systems have been implemented on digital computers. Apart from the fact that the expense of a computer have gone down significantly with the advent of mass-produced microprocessors and microcontrollers, there are other contributing factors, such as, any changes or tuning are easily achieved with software, implementation is not susceptible to changes due to external conditions, data can be easily transferred to any distant location without delay, and exponential growth in the speed of the microcontrollers [2].

Digital computers work with sequences of numbers rather than continuous functions of time. Information is read and updated at discrete points in time – referred to as *sampling*. For instance, a radar tracking system provides information on an airplane's position and motion to a digital processor at discrete periods of time. When there is no such inherent sampling, an analog-to-digital (A/D) converter must be incorporated in a digital control system. The output of the controller is later converted from discrete form into an analog signal by a digital-to-analog (D/A) converter. A block diagram of a digital control system is shown in Figure 2.11.

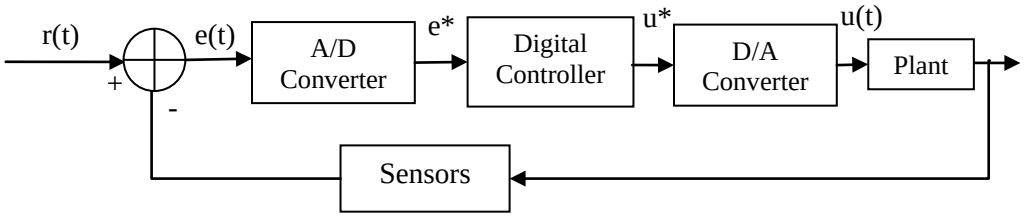


Figure 2.11: Digital control system

The notations *e** and *u** denote that these signals are sampled at specific time intervals. The extent of the information loss due to sampling depends on the sampling method and the related parameters. If a sequence of samples is taken of a signal *f(t)* every Δ seconds, then the sampling frequency needs to be large enough in comparison with the maximum rate of change of *f(t)*. Otherwise high frequency components will be mistakenly interpreted as low frequencies in the sampled sequence, a phenomenon known as *aliasing*.

For example, let us consider a signal $f(t) = 3\cos 2\pi t + \cos(20\pi t + \frac{\pi}{3})$. If the sampling period Δ is chosen to be 0.1 sec, then

$$f(k\Delta) = 3\cos(0.2k\pi) + \cos(2k\pi + \frac{\pi}{3}) = 3\cos(0.2k\pi) + 0.5 \qquad (2.12)$$

which illustrates the fact that the high frequency component has been shifted to a constant.

The output of a digital controller is another sequence of numbers, which need to be converted back to continuous time functions before they can be applied to the plant. Usually, this is done by interpolating them into a staircase.

2.1.7 Z-Transform

The discrete equivalent of a differential equation is called a *difference equation*. A linear high-order difference equation is of the form [4]:

$$y[k+n]+a_{n-1}y[k+n-1]+...+a_0y[k]=b_{n-1}u[k+n-1]+...+b_0u[k] \tag{2.13}$$

The Z-transform is the discrete equivalent of the Laplace transform. It is used to turn the difference equations into algebraic ones. For a sequence $\{y[k]; k = 0, 1, 2, \ldots\}$ the Z-transform pair is given by [4]:

$$Z[y[k]]=Y(z)=\sum_{k=0}^{\infty} z^{-k}y[k] \tag{2.14}$$

$$Z^{-1}[Y(z)]=y[k]=\frac{1}{2\pi j}\oint z^{k-1}Y(z)dz,\ j=\sqrt{-1} \tag{2.15}$$

where $Z(.)$ indicates the Z-transform operation and $Z^{-1}(.)$ represents the inverse Z-transform. The function *Y(z)* is essentially a power series in z^{-k} with coefficients equal to the values of the number sequences $\{y(k)\}$ [6].

2.1.8 Discrete Transfer Functions

If we take Z-transform on each side of the high-order difference equation (4) with zero initial condition, $A_q(z)\ Y_q(z) = B_q(z)\ U_q(z)$, where $Y_q(z)$ is the Z-transform of the sequences $\{y[k]\}$, $U_q(z)$ is the Z-transform of the sequences $\{u[k]\}$, $A_q(z) = z^n + a_{n-1}\ z^{n-1} + \ldots + a_0$, and $B_q(z) = b_m z^m + b_{m-1}\ z^{m-1} + \ldots + b_0$.

So, $Y_q(z) = G_q(z)\ U_q(z)$, where $G_q(z)=\frac{B_q(z)}{A_q(z)}$ and $G_q(z)$ is the discrete transfer function. As in the continuous-time case, the transfer function uniquely determines the input-output behavior at the discrete sampling times. However, the input-output behavior at times other than the sampling instant is undefined.

2.1.9 Continuous-to-Discrete Transfer Function Transformation

This transformation refers to turning the transfer function from the *s*-domain to the *z*-domain. There are different approaches to achieve this transformation, but two are most commonly used: bilinear transform and backward rule approach [36]. Tustin or bilinear transformation is a first-order approximation of the natural logarithm function that is an exact mapping of the *s*-domain to the *z*-domain. Since $z = e^{st}$, where T is the sample time or the reciprocal of the sampling frequency,

$$s = \frac{1}{T}\ln(z) = \frac{2}{T}[\frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \frac{1}{5}\left(\frac{z-1}{z+1}\right)^5 + ...] \approx \frac{2}{T}\frac{z-1}{z+1} \tag{2.16}$$

Thus the bilinear transform essentially uses first-order approximation and substitutes into the continuous-time transfer function as $s \leftarrow \frac{2}{T}\frac{z-1}{z+1}$. Bilinear transform gives a one-to-one mapping between analog frequency axis $s = j\omega_a$ and the digital frequency axis $z = e^{j\omega_d T}$, where T is the sampling interval. Thus the amplitude response is exactly the same on both axes, while the only defect being a *frequency warping* such that equal increments along the unit circle in the *z*-plane correspond to larger and larger bandwidths along the $j\omega$ axis in the *s*-plane [37].

The backward rule approach to continuous-to-discrete mapping stems from the definition of the derivative. The backward difference form of the derivative is [2]:

$$\frac{de}{dt} = \frac{e(t) - e(t-T)}{T} \tag{2.17}$$

Taking the Laplace transform, assuming all the initial conditions being zero,

$$sE(s) = \frac{E(s) - E(s)e^{-sT}}{T}$$

Or, $s = \frac{1 - e^{-sT}}{T}$ (2.18)

$$\text{Since } z = e^{sT},\ s = \frac{1 - z^{-1}}{T} = \frac{z - 1}{Tz} \tag{2.19}$$

Like the bilinear transform, the backward rule approach preserves stability (discussed in the next subsection) of the system and is free of aliasing. However, backward rule approximation requires that the sampling interval be small for accurate conversion [36].

2.1.10 Stability

A dynamic system is said to be *stable* if the output eventually reaches a finite steady-state value after an input or a disturbance. When the output oscillates with ever-increasing amplitude or it increases or decreases unidirectionally and without limit, the system is *unstable*. Let us consider a typical second-order system equation of the form [31]:

$$(a_2 D^2 + a_1 D + a_0)x = f(D)y \tag{2.20}$$

The transient response, and hence stability, of such system depends on the coefficients a_0, a_1 and a_2. If all the coefficients are above zero, the function will not contain any positive time exponentials and the system will be stable. However, if either a_1 or a_2 is less than zero, the transient response will contain positive exponentials and the system will be unstable. Figure 2.12 shows some stable and unstable responses [31].

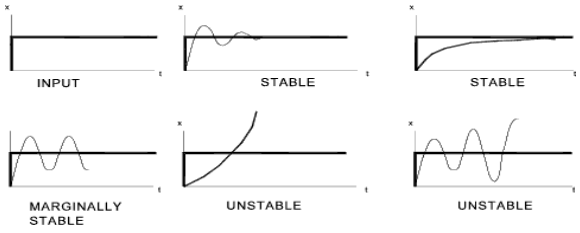


Figure 2.12: Examples of stable and unstable responses

Routh-Hurwitz stability conditions determine whether a system is stable or unstable. Let us consider the generalized equation [31]:

$$(a_n D^n + a_{n-1} D^{n-1} + ... + a_1 D + a_0)x = f(D)y \qquad (2.21)$$

Assuming a_0 is positive; a matrix is created of the coefficients:

$$\begin{array}{cccccccccc}
a_1 & a_0 & 0 & 0 & 0 & 0 & 0 & . & . & . \\
a_3 & a_2 & a_1 & a_0 & 0 & 0 & 0 & . & . & . \\
a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 & . & . & . \\
a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & . & . \\
a_9 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0
\end{array}$$

For stability of an equation of degree 4, the necessary conditions are as follows:

1. $a_1 > 0$, $a_2 > 0$, $a_3 > 0$, $a_4 > 0$.
2. $\begin{bmatrix} a_1 & a_0 \\ a_3 & a_2 \end{bmatrix} > 0$, that is, $a_1 a_2 - a_0 a_3 > 0$
3. $\begin{bmatrix} a_1 & a_0 & 0 \\ a_3 & a_2 & a_1 \\ 0 & a_4 & a_3 \end{bmatrix} > 0$, that is, $(a_1(a_2 a_3 - a_1 a_4) - a_0(a_3 a_3 - a_1.0)) > 0$

Any fourth-order equation has to meet these criteria to be stable. Note that [.] refers to the *determinant* of a square matrix.

2.2 Fuzzy Logic and Fuzzy Control

Fuzzy logic was originally proposed in theory by Zadeh [9][10], and implemented in automatic control by Mamdani [11][12]. Fuzzy controllers are a particular class of intelligent controllers built on fuzzy expert systems. One of the advantages of fuzzy controllers is that they do not require exact mathematical models unlike the traditional controllers, therefore making it easier to create highly nonlinear controllers in a very intuitive fashion. This property is very important for today's high-complexity and large networks where a precise mathematical model is often unavailable.

A fuzzy controller tries to mimic a knowledgeable human operator by applying a collection of control rules that might be overlapping and contradictory. The fuzzifier module receives data from sensor(s) as inputs and converts it by performing a simple mapping of a numeric quantity to a fuzzy set. The rulebase is formed by a collection of logical rules depicting the relationship between the input (antecedent) and output (consequent) of the controller [25]. The approximate reasoning algorithm is implemented in the inference engine whose task is to determine the firing degree of each rule, and to produce a fuzzy output by performing a weighted composition of the consequents for each rule. Finally, a module called a defuzzifier converts the fuzzy outputs to a single crisp control signal to be sent to the process.

2.2.1 Basic Concepts of Fuzzy Logic

The underlying mathematical construct of fuzzy logic is a fuzzy set, which is a generalization of mathematical set theory [13]. It is a set having a characteristic function with a co-domain consisting of the unit interval [0, 1] rather than the usual discrete set {0, 1}, and the characteristic function is known as membership function. This allows for a gradual transition from having a certain property to not having it [14].

The concepts of fuzzy sets are highly intuitive, since they work the same way humans tend to reason. In the words of Zadeh [1]:

“Clearly the ‘class of all real numbers that are much greater than 1,’ or ‘the class of beautiful women,’ or ‘the class of tall men,’ do not constitute classes or sets in the usual mathematical sense of these terms.”

For example, let us consider a temperature of 20°C as *comfortable* [7]. In terms of mathematical set theory, anything less than 20°C would then be *not comfortable*, meaning *cold*, and anything more than 20°C would again be *not comfortable*, meaning *warm*. This makes a value of 19°C to be *cold*, which is counter-intuitive. From human perspective, it is still *quite comfortable*, which is the essence behind the idea of fuzzy sets. Likewise, the temperature value of 25°C is partially

compatible with the terms *comfortable* and *warm*, but totally incompatible with *cold*. This notion of fuzzy sets is shown in Figure 2.13, where temperature values up to 10°C are only *cold* and on or above 30°C are only *warm*. As the temperature rises from 10°C to 20°C, the degree of *cold* decreases while that of *comfortable* increases. This goes on till 20°C where degree of *comfortable* is 1 and there is no *cold*. Similarly, the degree of *warm* goes higher starting from 20°C and *comfortable* keeps going down. At 30°C, temperature is totally *warm* and not *comfortable* at all.

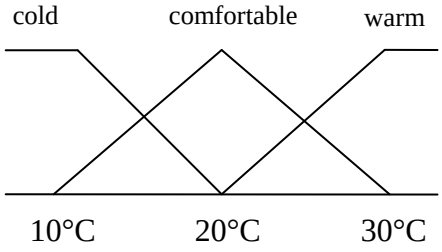


Figure 2.13: Notion of fuzzy sets

2.2.2 Membership Functions

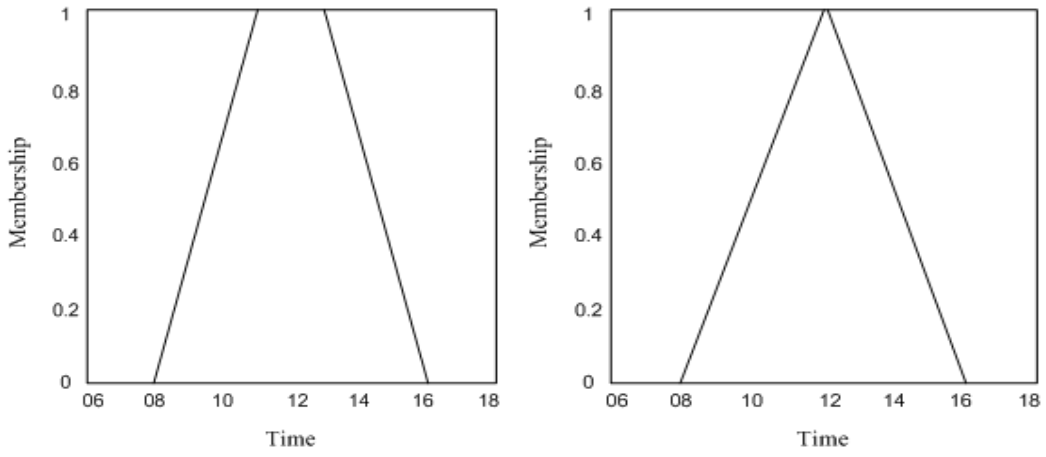
Membership functions are mappings from the universe of discourse to the unit interval [7]. There are two different ways to represent a membership function: continuous and discrete [8]. For instance, a trapezoidal membership function is a piecewise linear, continuous function, and it is based on four parameters {*a*, *b*, *c*, *d*} [15].

$$\mu_{Trapezoid}(x;a,b,c,d)=\left\{\begin{array}{l}0, x\le a\\ \dfrac{x-a}{b-a}, a\le x\le b\\ 1, b\le x\le c\\ \dfrac{d-x}{d-c}, c\le x\le d\\ 0, d\le x\end{array}\right\}, x\in\Re$$

A triangular membership function is piecewise linear, and derived from the trapezoidal membership function by setting *b* = *c*. Figure 2.14 shows the

trapezoidal and triangular membership functions representing the time ‘around noon’ [8].

Membership functions take many forms such as triangular, exponential, Gaussian membership functions and so on [7]. A discrete fuzzy set is defined by ordered pairs $\{(x_1, \mu(x_1)), (x_2, \mu(x_2)), \ldots\}$ where $\mu(x_i)$ is an evaluation of the membership function μ at a discrete point x_i [8].



(a) (b)

Figure 2.14: (a) Trapezoidal and (b) triangular membership functions.

2.2.3 Fuzzy Set Operations

Fuzzy set operations, namely – *intersection, union* and *complement*, are defined as functions of membership values. Functions that qualify as fuzzy intersections and fuzzy unions are usually referred to in the literature as *t-norms* and *t-conorms*, respectively.

2.2.3.1 Fuzzy Intersection

The intersection of two fuzzy sets *A* and *B* is defined by a binary operation on the unit interval; that is, a function of the form

$$i : [0,1] \times [0,1] \rightarrow [0,1]$$

For each element *x* of the universal set, this function takes as its argument the pair consisting of the element's membership grades in set *A* and in set *B*, and yield the membership grade of the element in the set constituting the intersection of *A* and *B*. Thus,

$(A \cap B)(x) = i[A(x), B(x)]$ for all $x \in X$.

Functions known as *t*-norms possess properties which ensure that fuzzy sets produced by *i* are intuitively acceptable as meaningful fuzzy intersections of any given pair of fuzzy sets [21]. Therefore, the class of *t*-norms is now generally accepted as equivalent to the class of fuzzy intersections.

A fuzzy intersection or t-norm *i* is a binary operation on the unit interval that satisfies at least the following axioms [21] for all $a, b, d \in [0,1]$:

Axiom I1: $i(a, 1) = a$ (Boundary condition)

Axiom I2: $b \leq d$ implies $i(a, b) \leq i(a, d)$ (Monotonicity)

Axiom I3: $i(a, b) = i(b, a)$ (Commutativity)

Axiom I4: $i(a, i(b, d)) = i(i(a, b), d)$ (Associativity)

Axiom I5: *i* is a continuous function (Continuity)

Axiom I6: $i(a, a) \leq a$ (Subidempotency)

The following are some examples of *t*-norms used as fuzzy intersections:

Standard Intersection: $\mu_{A \cap B} = \min(\mu_A, \mu_B)$

Algebraic Product: $\mu_{A \cap B} = \mu_A * \mu_B$

Bounded Difference: $\mu_{A \cap B} = \max(0, \mu_A + \mu_B - 1)$

Drastic Intersection: $\mu_{A \cap B} = \mu_A$ when $\mu_B = 1$

μ_B when $\mu_A = 1$

0 otherwise

2.2.3.2 Fuzzy Union

The general fuzzy union of two fuzzy sets *A* and *B* is specified by a function

$$u : [0,1] \times [0,1] \rightarrow [0,1]$$

The function returns the membership grade of the element in the set $A \cup B$. Thus,

$(A \cup B)(x) = u[A(x), B(x)]$ for all $x \in X$.

The properties of functions known as *t*-conorms are exactly the same as those of a function *u* that is acceptable as a fuzzy union; therefore, *t*-conorms and fuzzy unions are used interchangeably.

A fuzzy union or *t*-conorm *u* is a binary operation on the unit interval that satisfies at least the following axioms for all $a, b, d \in [0,1]$:

Axiom U1: $u(a, 0) = a$ (boundary condition)

Axiom U2: $b \leq d$ implies $u(a, b) \leq u(a, d)$ (monotonicity)

Axiom U3: $u(a, b) = u(b, a)$ (Commutativity)

Axiom U4: $u(a, u(b, d)) = u(u(a, b), d)$ (Associativity)

Axiom U5: *u* is a continuous function (Continuity)

Axiom U6: $u(a, a) \geq a$ (Superidempotency)

Axiom U7: $a_1 < a_2$ and $b_1 < b_2$ implies $u(a_1, b_1) < u(a_2, b_2)$ (Strict monotonicity)

Some examples of *t*-conorms used as fuzzy unions are given below:

Standard Union: $\mu_{A \cup B} = \max(\mu_A, \mu_B)$

Algebraic Sum: $\mu_{A \cup B} = \mu_A + \mu_B - \mu_A \mu_B$

Bounded Sum: $\mu_{A \cup B} = \min(1, \mu_A + \mu_B)$

Drastic Union: $\mu_{A\cup B} = \mu_A$ when $\mu_B = 0$

μ_B when $\mu_A = 0$

1 otherwise

2.2.3.3 Fuzzy Complement

Let *A* be a fuzzy set on *X*. Therefore, *A(x)* is interpreted as *the degree to which x belongs to A*. Let *cA* be a fuzzy *complement* of *A*. Then, *cA(x)* may be interpreted not only as the degree to which *x* belongs to *cA*, but also as *the degree to which x does not belong to A*.

Let a complement *cA* be defined by a function $c:[0,1]\rightarrow[0,1]$ which assigns a value *c(A(x))* to each membership grade *A(x)* of any given fuzzy set *A*. The value *c(A(x))* is interpreted as the value of *cA(x)*. Given a fuzzy set *A*, we obtain *cA* by applying function *c* to values *A(x)* for all $x \in X$.

To produce meaningful fuzzy complements, function *c* must satisfy at least the following two axioms:

Axiom C1: $c(0) = 1$ and $c(1) = 0$ (Boundary condition)

Axiom C2: For all $a, b \in [0, 1]$, if $a \leq b$, then $c(a) \geq c(b)$ (Monotonicity)

The complement of a fuzzy set *A* with membership function μ_A usually corresponds to the connective NOT, and has the membership function

$\mu_{\bar{A}} = 1 - \mu_A$.

2.2.4 Fuzzy Logic

The ‘truth’ or ‘falsehood’ assigned to a proposition is its truth-value [8]. For two-valued logic, truth-values can only be either 0 (false) or 1 (true). Fuzzy logic is an extension of the range of truth-values to the continuous interval [0, 1] of real numbers [16]. In mathematics, the word ‘and’ is used to join two sentences to

form the *conjunction* of the two sentences. The word ‘or’ is used to form the *disjunction* of two sentences. From two sentences, we may construct one, of the form ‘If…then…’, called an *implication*. The sentence following ‘If’ is the *antecedent*, and the sentence following ‘then’ is the *consequent*. A sentence that is modified by the word ‘not’ is called the *negation* of the original sentence. The words ‘and’, ‘or’, ‘If-then’ are called *connectives*.

Fuzzy logic connectives are defined similarly. If p is a fuzzy set, negation is defined as set complement, that is,

$$\neg p = 1 - p$$

Disjunction is defined as set union, that is, for fuzzy sets p and q

$$p \vee q \equiv \max(p, q)$$

Conjunction is defined as set intersection, that is, for sets p and q

$$p \wedge q \equiv \min(p, q)$$

Rules of inference specify conclusions drawn from assertions known or assumed to be true. One such rule of inference is *modus ponens*. It is often presented in the form of an argument:

$$\begin{array}{l} P \\ P \Rightarrow Q \\ \_\_\_\_\_ \\ Q \end{array}$$

It means, if P is known to be true, and we assume that $P \Rightarrow Q$ is true, then Q must be true. The assertion P is the *premise*, the assertion $P \Rightarrow Q$ is the *implication*, and the last assertion is the *conclusion*.

Modus ponens generalized to fuzzy logic is the core of fuzzy reasoning. Let us consider the argument [8]:

$$A'$$
$$A => B$$
$$-----$$
$$B'$$

It is similar to modus ponens, but the premise A' is slightly different from A and thus the conclusion B' is slightly different from B. For example [100],

This tomato is very red

If a tomato is red, then the tomato is ripe

This tomato is very ripe

This is generalized modus ponens.

A fuzzy rule has the form: 'If x is A then y is B', in which A and B are fuzzy sets. This is an implication where 'x is A' is the antecedent and 'y is B' is the consequent.

2.2.5 Fuzzy Controller

A fuzzy control system has a similar architecture to a conventional feedback control system. Figure 2.15 shows a fuzzy controller.

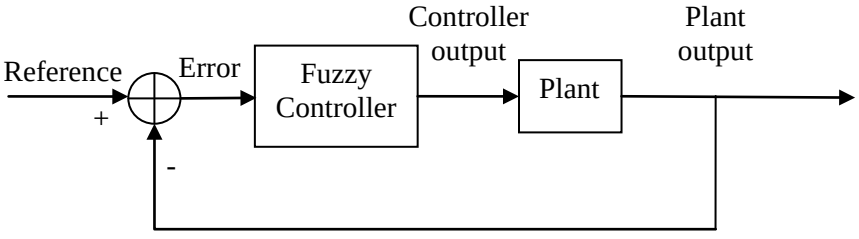


Figure 2.15: Fuzzy control system [8]

The difference between a fuzzy controller and its conventional counterpart lies in the control strategy. While the traditional system applies mathematical models using differential equations to calculate the controller output, a fuzzy controller

processes the inputs based on rules expressed in a more or less natural language. The advantage of using rule-based fuzzy controller stems from the fact that mathematical models of many control processes may not exist, or may be mathematically intractable [17].

2.2.6 Components of Fuzzy Controller

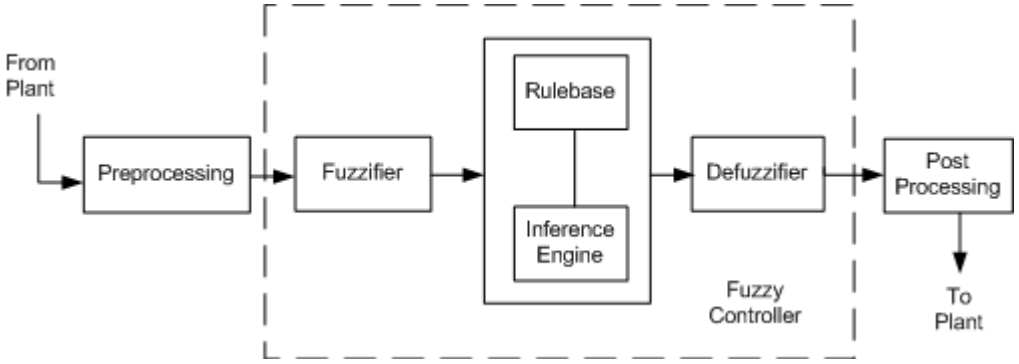


Figure 2.16: Fuzzy controller block diagram [8]

Figure 2.16 shows the general structure of a fuzzy controller. The preprocessor prepares usable data for further processing from measured inputs by the measuring device. Preprocessing may involve quantization in connection with sampling or rounding to integers, normalization or scaling, filtering to eliminate noise, averaging to obtain tendencies, and so on [8].

2.2.6.1 Fuzzifier Unit

The fuzzifier module calculates the membership grades, expressed in real numbers, from the membership functions. It evaluates the input measurements according to the premises of the rules. Each premise produces a membership grade expressing the degree of fulfillment of the premise. A lookup table [18] usually contains the membership values for all possible numerical inputs. This approach, albeit requiring more memory space, can be much faster than calculating the membership values in real-time [19].

2.2.6.2 Rule Base Unit

Three distinct variants of the fuzzy rule base have evolved so far – Mamdani, TSK or Sugeno and Tsukamoto rule bases. They use the same general inference scheme, but differ with respect to the conclusion membership functions.

Mamdani rule base uses fuzzy sets as consequents; therefore Mamdani controllers are computationally more expensive, although the implementation is more intuitive and well suited to human inputs. TSK or Sugeno rule base replaces the consequents of Mamdani model with a polynomial equation (usually either a 0^{th} or 1^{st} degree polynomial) of the (non-fuzzified) input variables. The rules take the following form:

If x is A and y is B then $z = f(x, y)$

where A and B are fuzzy sets in the antecedent and $z = f(x, y)$ is a crisp polynomial function in the consequent. For a zero-order Sugeno model, z is a constant. It is computationally efficient, works well with linear techniques, has guaranteed continuity of the output surface and well suited to mathematical analysis.

In the Tsukamoto rule base, the consequents are represented by fuzzy sets with monotonic membership functions. The rules take the form:

If x is A and y is B then $z = p*x + q*y + r$

Scalar values are often used as consequents in practical fuzzy controllers, referred as *singleton consequents*. Some of the benefits of using singleton consequents are simpler calculations, possibility of setting extreme values for the control signal and so on [8].

2.2.6.3 Inference Engine Unit

For each rule, the inference engine looks up the membership value where each input intersects a membership function. The *firing strength* α_i of a rule i is the degree of fulfilment of the rule premise. Rule i causes a fuzzy membership value corresponding to each input, which are aggregated using the ‘and’ or ‘or’ connective. The *activation* of a rule is the derivation of a conclusion depending on

the firing strength. Minimum or multiplication is used as the *activation operator* to activate a portion of each singleton consequent. If consequents are fuzzy sets as in the Mamdani controller, multiplication scales the membership curves while minimum clips them. However, for singletons, both operators result the same. All activated conclusions are *accumulated* using maximum or summation operator. The conclusions may contain several control actions.

2.2.6.4 Defuzzification Unit

The resulting fuzzy set from the inference engine has to be converted in the *defuzzifier* to a single number to form a control signal to the plant. There are dozens of defuzzification schemes, such as Center of Gravity (COG), Mean of Maxima (MOM), Bisector of Area (BOA) and so on [20], [90], [93], [96]. The crisp control value u_{COG} is the abscissa of the center of gravity of the fuzzy set. For singleton consequents, it takes the following form:

$$u_{COGS} = \frac{\sum_i y_i w_i}{\sum_i y_i} \tag{2.22}$$

Where y_i is the firing strength of each rule, and w_i is the consequent output. The term COGS stands for Center of Gravity for Singletons [8]. For fuzzy sets in the consequents, summations are replaced by integrals. It is a widely used method, but it is also computationally expensive.

The Bisector of Area (BOA) method finds the abscissa x of the vertical line that partitions the area under the membership function into two areas of equal size. For singleton consequents, u_{BOA} is the abscissa that minimizes

$$\left| \sum_{i=1}^{j} \mu_c(x_i) - \sum_{i=j+1}^{i_{\max}} \mu_c(x_i) \right|, \quad 1 < j < i_{\max} \tag{2.23}$$

Here $i_{\max}$ is the index of the largest abscissa $x_{i_{\max}}$. Its computational complexity is also relatively high.

Another approach is to choose the point of the universe with the highest membership. Mean of maxima (MOM) is taken among several such points.

$$u_{MOM} = \frac{\sum_{i \in I} x_i}{I}, I = \{i \mid \mu_c(x_i) = \mu_{\max}\} \quad (2.24)$$

Where I is the crisp set of indices I where $\mu_c(x_i)$ reaches its maximum $\mu_{\max}$, and $|I|$ is the number of members of the set. Although this method is not concerned about the shape of the fuzzy set, computationally it is faster.

2.3 Hardware Implementations of Fuzzy Controllers

Although software-based approaches [55], [56] on general-purpose microprocessors are the most flexible and economical in developing fuzzy controllers, and satisfactory results have also been obtained in the industrial areas [54]; existing software fuzzy implementations result in slow operational speed due to the fact that general-purpose computers do not have hardware computational units to implement fuzzy operations directly, making it inadequate for the high-speed processing requirement of real-time control problems [53], [58]. Therefore massive research activities on dedicated fuzzy hardware implementation manipulating both analog and digital approaches have been going on since the 1980s. Analog systems have been preferred where power consumption and resource usage are the main design criteria; whereas precision and compatibility issues favor the digital technology [23]. A fuzzy controller having a processing speed of 10 MFLIPS and developed by OMRON Corporation is considered a basic performance standard [24], although proposed prototypes in the literature have speeds up to 40 to 50 MFLIPS [25].

2.3.1 Analog Implementations

Implementation of analog fuzzy controllers began with the work of Yamakawa and Miki [26] in 1986. They developed nine basic fuzzy logic circuits in current mode, containing the realization of functions, such as bounded difference, fuzzy

complement, fuzzy logic union (MAX), bounded sum, fuzzy logic intersection (MIN), implication, and so on.

Although current-mode circuits are not exceedingly sensitive to changes in supply voltages, the fan-out number is limited [27], meaning that problems would arise when the inputs must be distributed to many operational blocks, a usual scenario for fuzzy rule base implementation. To overcome this issue, a tunable voltage-input current-output membership function circuit is presented in [27] using 3.5 μm CMOS design rules based on the fact that voltage inputs can be easily distributed to many rule blocks.

N-input MIN/MAX circuits are used in the inference engine of a fuzzy controller. A current-mode multi-input MAX circuit is proposed by Baturone *et al.* in [32] and [35]. The MAX circuit also performs the MIN operation using De Morgan's law. It uses $3n + 1$ transistors while achieving the same level of precision with less area and power consumption compared to $5n + 1$ transistors in [28]. The MAX/MIN circuit design in [30] follows the similar structure as in [29], but it only requires $n + 4$ MOS transistors, which is an improvement over other implementations.

A current-mode defuzzification circuit based on the square-law MOS characteristic is described in [34] providing high linearity and large dynamic range. A square-root circuit is cascaded with a squarer circuit to implement the multiplier block. Addition is realized by wiring the outputs of the multiplier blocks together. As the use of this circuit is limited to current-mode only, a voltage-to-current (V-I) and current-to-voltage (I-V) conversions might be needed. The division in defuzzification is avoided in [28] by a Normalized Locked Loop (NLL). The denominator in the COG equation is made constant with a negative feedback loop, so it becomes a weighted sum operation instead of weighted average. The maximum simulated error is reported to be within 0.4% of the full scale current. A simple current-input voltage-output continuous-time divider circuit is implemented in [35] using a variable transresistance technique, where a resistance value is controlled by a current. It offers larger dynamic range

compared to the approach in [34] and high-frequency operation. The defuzzifier in [40] is implemented by a multiplier, a divider, three integrators, an I-V converter, an attenuator and a division control unit. To avoid the nonlinearity of the multiplier for signals greater than 0.5-V, the attenuator is used. An I-V converter is needed because the multiplier block is based on a voltage-mode integrator. A division control unit gets rid of very small signals to be applied to the denominator.

Programmable analog fuzzy chips have also been developed over the last couple of decades. Continuous, fuzzy, and multi-valued logic circuits are realized using a general-purpose Field Programmable Analog Array (FPAA) in [43]. The FPAA is based on an array of current-mode processing cells, operates from ±3.3 V or ±5 V power supplies, and works with frequencies up to several hundred MHz. In [44], a similar field programmable analog fuzzy processor is proposed supporting fifteen rules, three inputs and one output. The architecture is split into an analog core and a digital part allowing field programmability. The digital segment relies on a software tool to compute the programming values. The chip area is 32 mm^2 using a CMOS 0.7-μm n-well technology with the analog core occupying only 7%. The settling time for a step response is reported to be < 0.6 μs.

Miki and Yamakawa [46] proposed an analog fuzzy processor with an inference engine of over 1 MFLIPS excluding defuzzification. To have a flexible system configuration, inferences and defuzzification are handled by two separate chips. The executed rule can be changed dynamically from one rule set to the other stored in the on-chip rule memory. The chip in [47] occupies an area of 17.9 mm^2 and processes up to 131072 rules, 4096 inputs and 1024 outputs with different membership functions. Fuzzification of four 12-bit inputs, inference of 80 rules, and COG defuzzification for a 16-bit output takes 16 μs, translating to a processing speed of 62.5 KFLIPS. Baturone *et al.* [48] designed general-purpose fuzzy chips allowing fully-parallel analog rule processing and optimized digital circuitry for programmability. The analog core of a two-input processor occupies a silicon area of 1 mm^2 and the response time is less than 2 μs.

Although analog implementations have features such as lower power consumption [23], and not having the requirement of analog-to-digital (A/D) or digital-to-analog (D/A) converters to communicate with sensors and actuators, they suffer from poor flexibility, less precision due to noise, distortion, interferences and parameter mismatch [57], and compatibility issues with other digital systems [52].

2.3.2 Digital Implementations

There are many different ways to implement fuzzy control systems in digital hardware. One approach is the use of general-purpose processors with a number of instructions dedicated to fuzzy operations. While this method [85], [86] provides features such as flexibility and automatic support of non-fuzzy computations, performance is limited [84]. A better approach in terms of speed is special-purpose processors designed only for fuzzy operations [87], [88]. However, they have poor flexibility and cannot be used as stand-alone processors to implement the control system. Most of the studies so far, therefore, have dealt with dedicated hardware tailored to a particular fuzzy application [64]. Although this scheme is not flexible at all, the advantages include very fast processing capability required for real-time systems, low cost in terms of resources, and so on [84].

Togai and Watanabe [59] [60] were the first to implement fuzzy systems in digital hardware in the mid 1980s. They developed a VLSI chip, fabricated in 2.5-µm CMOS technology, to perform the fuzzy inference process. The rulebase is stored in Read-Only Memory (ROM) because of its faster operation and less area requirement compared to the Random Access Memory (RAM). The degree of membership functions is represented by 4 bits, so 16 discrete levels are permitted. Two circuits with serial processing capabilities are used to implement MAX and MIN operations. The chip was first demonstrated at AT&T Bell lab. It operates on a 20.8 MHz clock with a processing speed of 80 KFLIPS.

The second generation of a 1-µm CMOS VLSI chip has been designed by Watanabe *et al*. [61] in the early 1990s. The chip consists of 688,000 transistors out of which 476,000 are used for RAM memory. Either 51 rules with 4 inputs

and 2 outputs, or 102 rules with 2 inputs and 1 output can be implemented at a speed of over 150 KFLIPS. The universe of discourse of a fuzzy set is divided into 64 elements each being a 4-bit number, therefore occupying 256 bits of memory for fuzzification table lookup. Instead of dedicating 256 bits for each fuzzy set, the membership functions for all fuzzy sets are stored in one memory area in [63]. The representations of the membership functions for each fuzzy set are overlapped. The effectiveness of this approach depends on the overlap factor of the fuzzy sets. For a factor of 2, it only requires half of the memory space compared to that of [61]. To speed up the inference process, the defuzzification operations are pre-computed and partial results are stored for runtime access. To save memory, a different way to implement membership functions is described in [62]. It makes use of three memory spaces to store the fuzzification information. Each fuzzy set is assigned a 3-bit number in the first space; the other two spaces keep track of the values that the membership functions possess for any pair of active fuzzy sets for a particular input.

A custom designed hardware fuzzy logic controller with on-line adaptability is described in [65], [66]. Look-up tables are used for fuzzification with 4-bit resolution. To avoid division in defuzzification, all possible division outputs are pre-calculated and stored in a look-up table with concatenated numerator and denominator values being used as addresses to access the table contents. To facilitate on-line adaptation, six extra SRAMs (Static Random Access Memory) are used to store input membership functions along with 6 others for normal operations. A fully pipelined version can obtain a speed up to 9 MFLIPS. A similar design approach [58] achieves 3.3 MFLIPS operational speed, while a multilevel systolic approach [67] goes up to 10 MFLIPS with the use of 30,000 gates and 3 KB memory. An adaptive fuzzy controller oriented to ASICs is presented in [75]. Adaptation algorithm works by updating the parameters of membership functions and rules based on new input data. During each cycle, the rules with more firing strength are reinforced while others are ignored depending on the degree of their strength. The circuit is fabricated in 0.7 μm CMOS technology with 35,000 gates.

A general-purpose hardware system called KAFA (KAist Fuzzy Accelerator) to provide various fuzzy inference methods and fuzzy set operations is presented in [54], [69]. Fuzzy Processing Elements (FPEs) are separate from the main controller. Each FPE is connected to its next neighbor, while the first and last ones are connected to the main controller to form a ring topology. The first prototype of KAFA is implemented on an FPGA with 128 FPE's. An FPE unit consists of a Fuzzy Arithmetic Logic Unit (FALU), a memory space to store 8-bit membership values, a register file containing nine registers and a communication link. With 11 basic instructions, FALU implements 8 fuzzy set operations, namely – logical product, logical sum, algebraic product, algebraic sum, bounded product, bounded sum, drastic product and drastic sum. KAFA also provides 3 defuzzification methods – maximum criterion, mean-of-maxima, and modified center of area – with the first two dealing with the problem of finding the FPE containing the largest membership value in a list of FPEs. A similar hardware board named Future Board [72], [73] to process fuzzy set operations is proposed by Tokunaga. It consists of 4 fuzzy set processors (FSPs) to concurrently execute four streams of fuzzy operations on 8-bit data. Lee and Bien [74] developed a flexible fuzzy control system called FLEXi. For 8 inputs, 4 outputs and 256 rules with 16 MHz clock, it operates approximately at 20 KFLIPS.

Sánchez-Solano *et al.* [70] described two programmable fuzzy controllers that provide low resource usage and relatively high operational speed by adopting some restrictions on the degree of overlapping of antecedent membership functions and by using simplified defuzzification methods employing singleton consequents. Inference time is reduced by identifying active rules [71] defined as such rules for which all the antecedent membership grades are non-zero. The prototypes are implemented in a 1-µm CMOS technology, and they consist of 3 inputs and 1 output using 8 membership functions with an overlapping degree of 2. A memory space of 64 x 15 bits is used for membership functions of each input. They achieve an inference speed of 3 MFLIPS.

Ascia *et al.* [79] presented a VLSI fuzzy processor with 3 inputs, 1 output and 32 rules, which achieved 5.2 MFLIPS for a clock frequency of 66 MHz. A fuzzy processor [80] with four 7-bit inputs, one 7-bit output, 7 input membership functions, 8 output membership functions and 127 rules is implemented in 1 μm CMOS technology. It is used in a trigger system in High Energy Physics Experiments, and has a speed of 50 MFLIPS. It uses the active rule selector approach presented in [71].

An asynchronous computational approach to the hardware design of a fuzzy controller is described in [68]. The controller is designed as a large asynchronous pipeline in which most of the hardwired delays are replaced by self-timed combinational logic controlled by handshaking signals rather than a global clock. This scheme is useful for low-power embedded applications.

Due to the significant advances in the digital field during the last few decades, digital approach for fuzzy hardware realization is now easier to design, more precise, and more flexible in comparison to analog implementation [58].

2.3.3 Fast Defuzzification

Defuzzification has always been a bottleneck for faster implementation of fuzzy systems. Therefore a number of techniques have been proposed in the literature to avoid computationally expensive multiplication and division operations. In [89], a heuristic approach based on adapting any fuzzy output shape into one single triangle and estimating the centroid position is presented. The processing time for this approach is reported to be 23 times less than that for COG defuzzification. Runkler and Glesner [92] proposed a centroid approximation algorithm (CADE) to decrease the computation cost for defuzzification. Further reduction in computation is achieved by another algorithm (DECADE) which avoids multiplication and division operations involved in the previous method. The maximum error is reported to be about 7% for DECADE.

Eisele *et al.* [91] presented a fast defuzzification method for hardware implementation of fuzzy inference algorithms. This approach optimizes the COG

method by skipping all the regions for which the output possibility distribution is zero. As it adapts to the degree of contribution of the regions to the final crisp value, it is termed as the adaptive integration. One drawback of this method is that the implementation becomes complicated because of the extra circuitry needed to determine the relevance of the regions. A similar approach is followed by Tamukoh *et al.* [94], [95]. They proposed a bit-shifting-based fuzzy inference method which uses only the "active units", meaning those rules which have large influence on the defuzzified output. The concept of "active rules", rules that will contribute to the final result, is also used to speed up the defuzzification process in [98]. Two similar rule selector blocks are used to examine two rules at every clock cycle. The division operation is replaced by multiplication with the estimated reciprocal of the denominator.

Three different implementations of COG defuzzification are proposed in [99]. These methods compute the defuzzified output when the output membership functions are trapezoidal and form a fuzzy partition. The discretization method is the easiest among all to implement, because of the fact that the fuzzy output is approximated by a number of rectangles of equal width and height. With the increasing number of rectangles, the narrower they become and output is also better approximated. In the slope-based technique, fuzzy output surface is partitioned in such a way that the slope of the fuzzy output is constant within each part and different in two adjacent parts. The modified transformation function method originates from the transformation method presented in [97]. These methods are not as straightforward as discretization, but they allow faster and more accurate computation.

A COG method with only integer additions and one integer division is presented in [101]. The proposed algorithm maps the real values of fuzzy membership functions onto an integer grid. It is 12.75 times faster than conventional COG for the truck-backer problem. Introduction of quantization error is one disadvantage, because of the mapping of real values to integer ones. Although multiplication is eliminated, one division operation is still required.

CHAPTER 3
PROPOSED LOG-DOMAIN CONTROLLER

3.1 Logarithmic Arithmetic

Logarithmic-domain arithmetic has been used to accelerate hardware for many applications where there are a large number of multiplication and/or division operations involved, e.g. in [83]. In the fuzzy controller, the defuzzifier enjoys the greatest advantage of log-domain arithmetic, since the COG method in equation (2.22) requires three fundamental operations: multiplication, summation and division. If $x = q_1/q_2$, then $log(x) = log(q_1) - log(q_2)$, and hence a divider is replaced by a much simpler subtractor. Computationally expensive multiplication operations are similarly carried out by addition circuits in log-domain. The fundamental principle of this log-domain system is to take the logarithmic transform of all quantities. For instance, instead of performing a computation $x = f(q_1, q_2)$, the quantity $log(x)$ is computed using an equivalent function $g(log(q_1), log(q_2))$. If all quantities fall in the range (0, 1], all the logarithmic values are either 0 or a negative number, and the negative signs can be safely ignored simplifying calculations. As fuzzy membership values are always within (0, 1], (assuming we ignore membership values equal to 0 in our computations, as is commonly done) fuzzy logic controllers are suitable for this type of arithmetic.

Since we are using absolute values of all logarithmic quantities, minimum operations are changed to maximum and vice versa in the inference engine of the log-domain controller. However, the summation operation is complicated in the log-domain. If $x = q_1 + q_2$, with $q_1 > 0$ and $q_2 > 0$, then it can be shown [82]:

$$\log(x) = \log(q_1 + q_2) = \max(\log(q_1), \log(q_2)) + \log(1 + e^{-|\log(q_1) - \log(q_2)|}) \qquad (3.1)$$

The second logarithmic term in equation (3.1) becomes almost zero when q_1 and q_2 are not close in value. This term is often ignored in log-domain arithmetic at a small loss in precision [83], but can also be very crudely approximated by a

correction factor [81], [82] (discussed in the next section). We have implemented in simulation two versions of our log-domain controller – one without the correction factor, and the other with the factor.

3.2 Approximate Correction Factor

In this section we describe how a series of numbers are added in the log-domain controller. Say, we need to calculate the sum of a series of numbers $q_1, q_2, \ldots, q_n$, while the only information available to us is the set of their logarithmic values.

$$\text{Now, } \log(x) = \log(q_1 + q_2 + \ldots + q_n) = \log(e^{\log(q_1)} + e^{\log(q_2)} + \ldots + e^{\log(q_n)}) \qquad (3.2)$$

If q_1 is the maximum among all the q_i's, $log(q_1)$ is also the maximum among all the log values. Now, from (3.2),

$$\log(x) = \log(e^{\log(q_1)}(1 + e^{\log(q_2)-\log(q_1)} + \ldots + e^{\log(q_n)-\log(q_1)}))$$

$$= \log(q_1) + \log(1 + e^{\log(q_2)-\log(q_1)} + \ldots + e^{\log(q_n)-\log(q_1)}) \qquad (3.3)$$

If q_2 is the second largest value, the impact of $e^{\log(q_2)-\log(q_1)}$ is the highest among all the other terms. So in our approximation, we treat them as follows:

$$\log(1 + e^{\log(q_2)-\log(q_1)} + \ldots + e^{\log(q_n)-\log(q_1)}) \approx e^{\log(q_2)-\log(q_1)} \qquad (3.4)$$

This is the correction factor applied to our system. It involves determining two maximum (or, minimum) values instead of one. The advantage of using this correction factor is that the control surface becomes better in terms of RMS (root-mean-square) difference with respect to the typical fuzzy controller (discussed in the next chapter). However, step response curves for log-domain controllers with and without a correction factor show no significant difference. The drawback of the correction factor is the added computational complexity. Four addition/subtraction operations and two lookup tables are required to implement it.

3.3 Design of the Controller

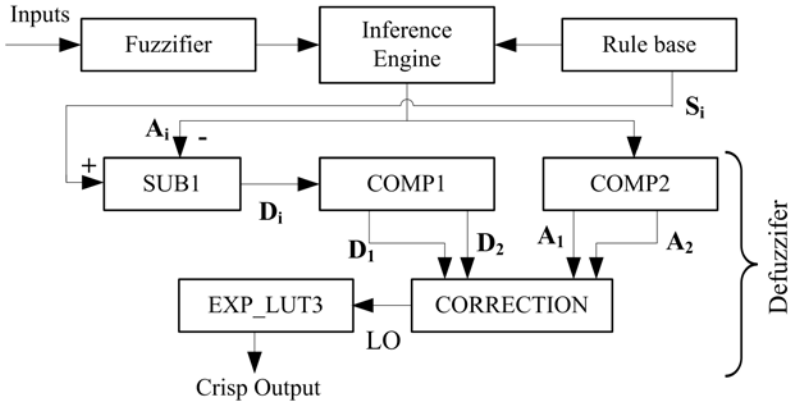


Figure 3.1: Block diagram of the log-domain controller

Figure 3.1 shows the block diagram of the log-domain controller. The inputs are first passed to the fuzzifier module. It consists of a lookup table (LUT) with *m+1* number of columns, where *m* is the number of membership functions. The first column of each row contains the possible values of the inputs within the input universe and the other columns have the logarithmic membership values. As all the log values are negative, we store the values without the signs. We note that storing logarithmic values in the fuzzifier LUT does not require any extra hardware compared to a conventional fuzzy LUT.

The inference engine module usually determines the minimum value among the antecedents for each rule. Since the negative antecedent values are stored as positive, we use maximum function instead of minimum, and the outputs (A_i) are passed to the subtractor block, SUB1, inside defuzzifier. The other input for SUB1, log of all consequent outputs (S_i), comes from the rulebase. This subtractor block and the minimum function inside COMP2 are used because of the positive A_i values. The subtracted outputs D_i go to COMP1 block which calculates the first two maximum values, *D1* and *D2*, respectively. The second maximum value is selected to be used later in the CORRECTION block. Similarly, at the same time, two minimum values, *A1* and *A2*, are chosen from A_i.

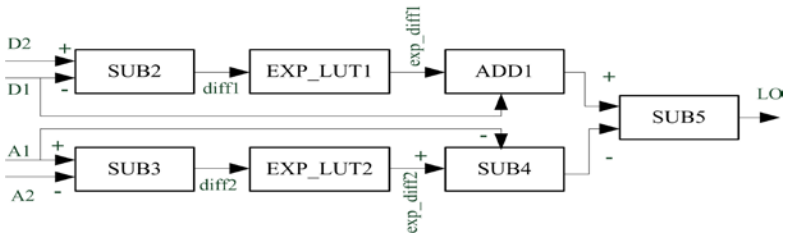


Figure 3.2: Block diagram of CORRECTION

Inside the CORRECTION block (see Figure 3.2), a subtractor block, SUB2, computes the difference between *D2* and *D1*, and passes to EXP_LUT1 to get the corresponding exponential value, *exp_diff1*. An adder block, ADD1, computes *term1* as *D1* plus exp_diff1. Similarly, a parallel subtractor block, SUB2, is used for *A1* and *A2*, and the output is sent to EXP_LUT2 to determine *exp_diff2*. Then *A1* is subtracted from *exp_diff2* to obtain *term2*. This subtraction effectively functions as an addition with the exception that *A1* is changed back to its original negative value. The final subtractor block, SUB5, calculates the difference, *LO*, between *term1* and *term2*, and passes it back to defuzzifier module for exponential calculation, which results in the final crisp output.

It is worth mentioning that if the approximate correction factor is ignored, the CORRECTION block only contains an adder block which sums *D1* with *A1*. This is a significant simplification with a negligible performance penalty.

3.4 Simulation Design

In this subsection, we discuss how the log-domain controllers (with and without the correction factor) along with a typical fuzzy controller are implemented on MATLAB Simulink.

3.4.1 Typical Fuzzy Controller

The MATLAB Simulink block diagram for the typical fuzzy control system is shown in Figure 3.3. The plant transfer function is taken from experiment 1, described in the next chapter.

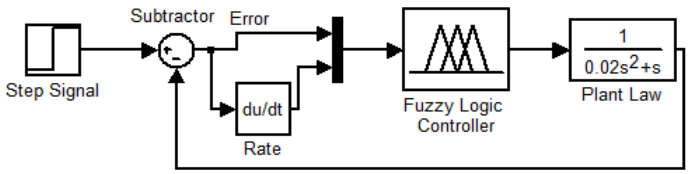


Figure 3.3: Simulink diagram for typical fuzzy controller

A step signal is passed to the Subtractor block at a specific sampling rate. The output from the plant is used as the negative input for Subtractor. For the first sample, the second input is considered to be zero. The subtracted value acts as the *error* input to the fuzzy controller while the derivative of error is the second input. The fuzzifier first converts the numerical inputs to fuzzy sets based on the corresponding membership functions. Note that for each input, there are at most two fuzzy sets which have non-zero contributions. This is because of the fact that the overlapping factor of membership functions is 2. Therefore there can be four "active" rules at a time. The inference engine calculates the firing strength of each rule by using the minimum function as AND connective of the antecedents. The defuzzifier multiplies the firing strengths with corresponding consequent outputs and then adds them together to complete the numerator of the COG equation. The denominator is calculated by summing the firing strengths, and finally the numerator value is divided by the denominator to get the crisp value. Note that multiplication and division operations in the defuzzifier result in a slow implementation, which is overcome by our log-domain controller.

3.4.2 Log-Domain Controller with Correction Factor

The Simulink block diagram for log-domain controller is shown in Figure 3.4.

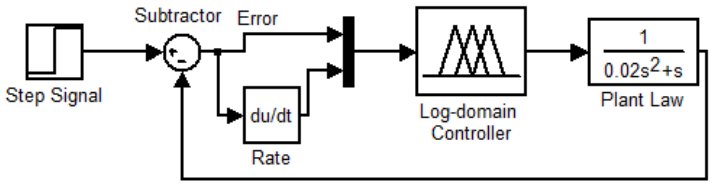


Figure 3.4: Simulink diagram of log-domain controller

As with the typical fuzzy controller, the fuzzifier receives the error and rate of error values as inputs. The fuzzifier then sends a set of logarithmic fuzzified values (log values of the membership degrees for each fuzzy set) for each input to the inference engine. Note that since all of the log values are 0 or negative, the sign is truncated and only absolute values are stored to simplify further calculation. The inference engine uses maximum function for the antecedents of each rule, because of the stored fuzzified values without signs. The rule base contains the log values of singleton consequents as well. The negative consequent values are treated as positive for logarithmic calculations. However, a separate array keeps track of all the signs of the consequents. This array is used later to adjust the sign of the final output.

The defuzzifier module first subtracts the log value of the firing strength of each rule from the log of corresponding consequent output. The subtracted result for all the rules are then passed to the comparator block which determines the largest two values (D_1 and D_2). At the same time, another comparator block computes the lowest two values (A_1 and A_2) from the log values of firing strengths. Inside the comparator block, a subtractor block calculates the difference between D_2 and D_1, and the exponential value of the result is obtained. The exponential value is added to D_1 to calculate *term1*. Similarly A_1 minus A_2 is performed and the exponential value is sent to a subtractor block. The other input of this block is A_1, and the result is *term2*. A subtraction block subtracts *term2* from *term1*, and sends the result back to the defuzzifier where the final result is set to its exponential value.

3.4.3 Log-Domain Controller without Correction Factor

The architecture of the controller without correction factor is the same as the one with the correction factor, with the exception of the correction block in the defuzzifier. The correction block is significantly simplified, because it only contains an adder block to calculate D_1 plus A_1. So instead of two, only one value (largest or smallest) is calculated in the comparators.

CHAPTER 4
EXPERIMENTAL RESULTS IN SIMULATION

Four controllers – typical fuzzy controller, PD controller, log-domain controller without a correction factor and with a correction factor, – are used in MATLAB simulation for our experiments.

4.1 Experiment 1

We compare and analyze the performance of the log-domain controllers (with and without correction factor) with a typical fuzzy and a PD controller. The benchmark controllers and the plant are taken from [78]. The transfer function of the second-order plant (representing a DC servomotor) used in our experiments is $\frac{1}{0.02s^2 + s}$. The output of the linear PD controller (shown in Figure 4.1) is calculated as $u = k_p e + k_d \frac{de}{dt}$, where e is the error and $\frac{de}{dt}$ is the rate of change of error.

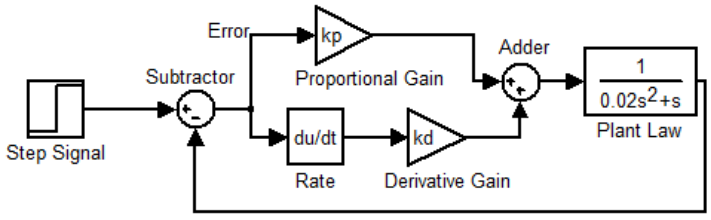


Figure 4.1: PD controller

The *error* and *rate* values are multiplied by a proportional (k_p) and a derivative (k_d) gain, respectively. Both the values are added together to send to the plant whose output is fed back to the Subtractor.

We measure the performance of our controllers with the help of step response curves and control surface plots. Rise time, settling time and overshoot – three parameters of the step response – are compared for different controllers. Next, we use the Root-Mean-Square (RMS) difference among the control surfaces to analyze the control laws.

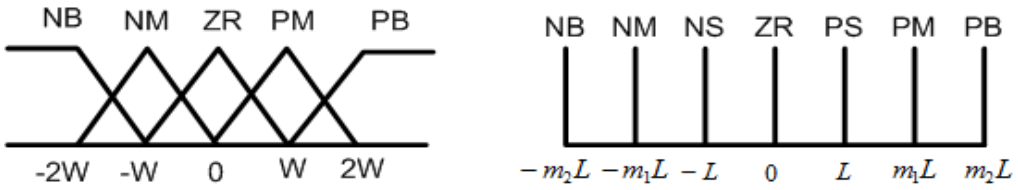
4.1.1 Controllers

The rule base for the fuzzy controllers is shown in Table 4.1.

Table 4.1: Rule base for the fuzzy controllers. Note that NB = Negative Big, NM = Negative Medium, ZR = Zero, PM = Positive Medium, and PB = Positive Big.

Input2/Input1	NB	NM	ZR	PM	PB
PB	ZR	PS	PM	PB	PB
PM	NS	ZR	PS	PM	PB
ZR	NM	NS	ZR	PS	PM
NM	NB	NM	NS	ZR	PS
NB	NB	NB	NM	NS	ZR

Input and output membership functions are shown in Figure 4.2.



(a) (b)

Figure 4.2: (a) Input and (b) output membership functions

The parameters (taken from [78]) for the fuzzy controllers are: $W_{input1} = 1$, $W_{input2} = 50$, $L = 50$, $m_1 = 2$, $m_2 = 3$, while those for the linear PD controller are: $P = 50$, $k_d = 1$.

4.1.2 Results

A comparison of the step response of the different controllers is presented in Figure 4.3. Table 4.2 illustrates the rise time, settling time, and overshoot for different controllers. *Rise time* is defined as the time the plant outputs take to get to 90% of the step size from a value of 10%. *Settling time* is the time for outputs to settle down within 2.5% of the steady state from control start. We express *overshoot* (output exceeds the steady state value) as a percentage relative to the final value of the plant output.

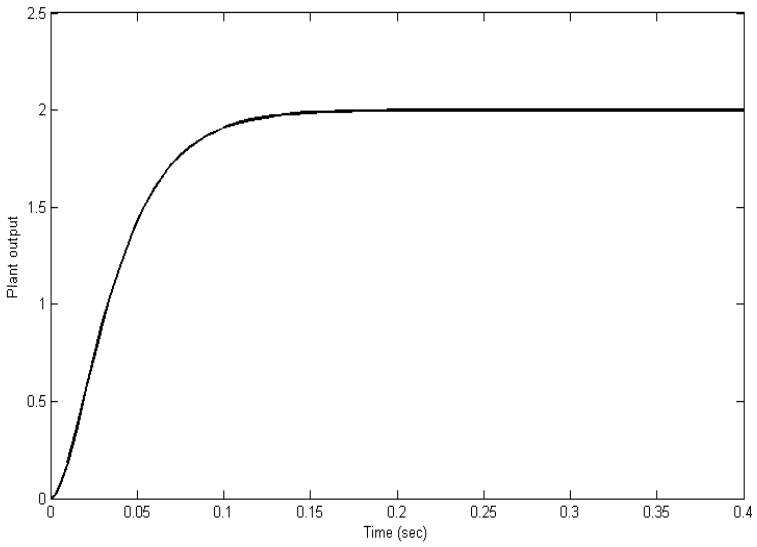


Figure 4.3 (a)

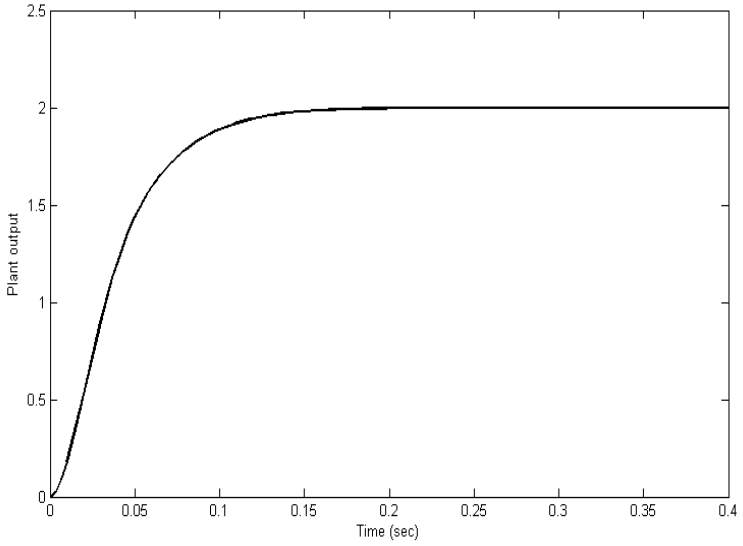


Figure 4.3 (b)

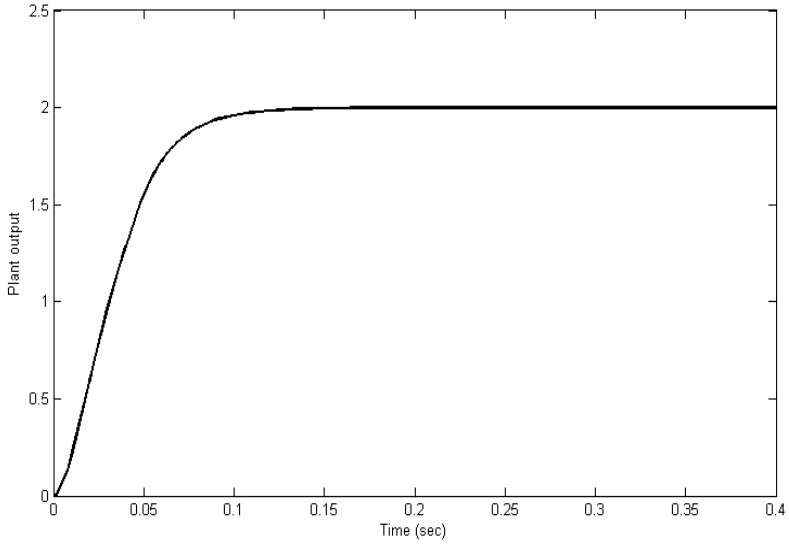


Figure 4.3 (c)

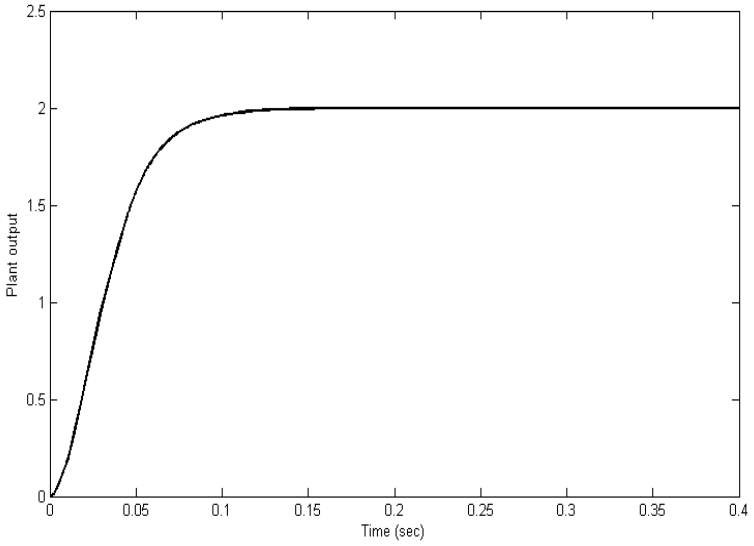


Figure 4.3 (d)

Figure 4.3: Step responses of different controllers – (a) linear PD, (b) typical fuzzy, (c) log-domain without a correction factor and (d) log-domain with a correction factor

Table 4.2: Rise time, settling time, and overshoot for different controllers with a sampling period of 0.01 s

Controllers	Rise Time	Settling Time	Overshoot (%)
PD	0.0688	0.1219	0
Typical fuzzy	0.0719	0.1298	0
Log-domain (uncorrected)	0.0567	0.1	0.0009
Log-domain (corrected)	0.0548	0.0997	0.0025

Figure 4.4 shows the control surface outputs generated for a linear PD controller, a typical fuzzy-logic controller, and log-domain controllers (with and without a correction factor). Table 4.3 reports the RMS difference between control surfaces of the controllers.

Table 4.3: RMS difference between control surfaces

Controllers	PD	Log-domain (uncorrected)	Log-domain (corrected)
Typical	31.8991	11.7040	5.2965
PD		32.2829	32.6604
Log-domain (uncorrected)			16.5292

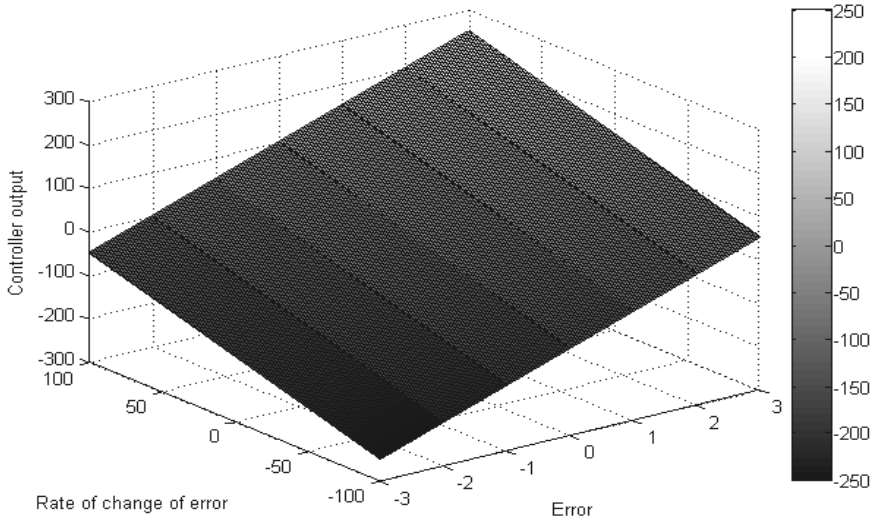


Figure 4.4 (a)

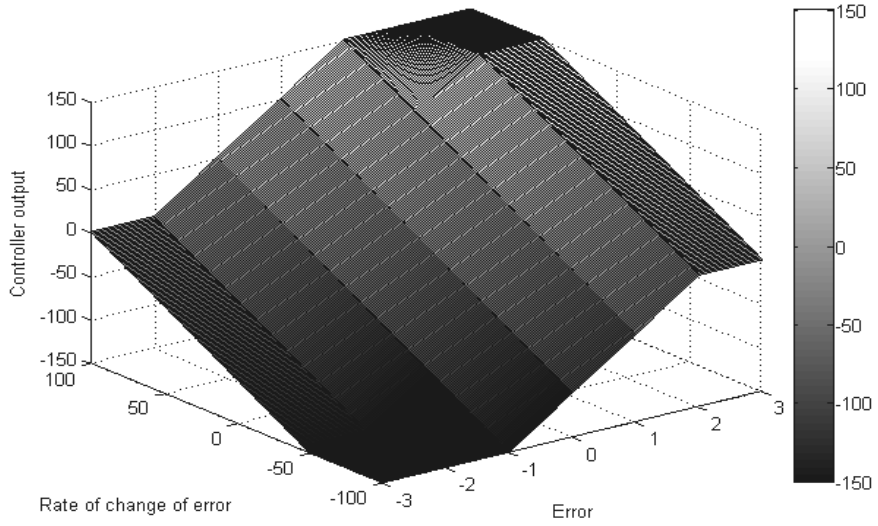


Figure 4.4 (b)

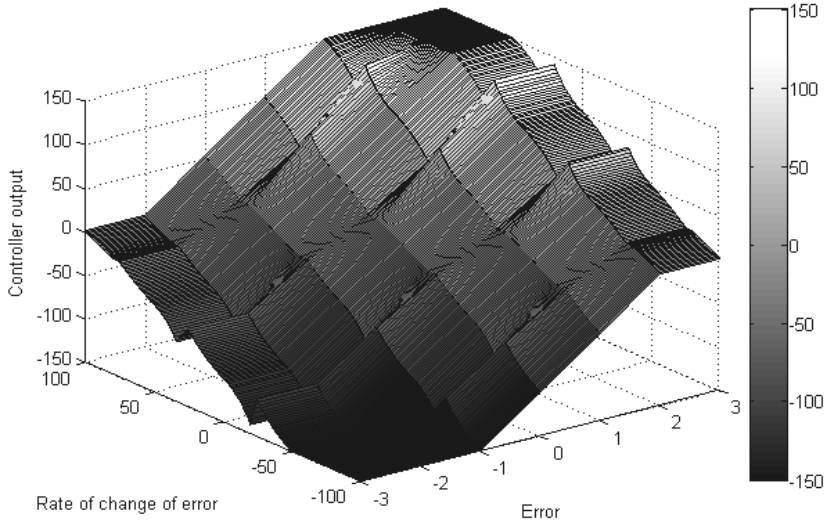


Figure 4.4 (c)

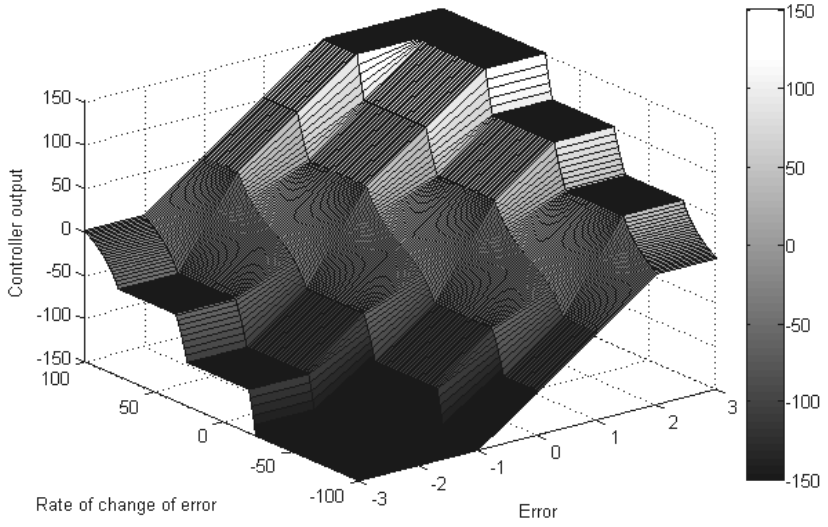


Figure 4.4 (d)

Figure 4.4: control surfaces of different controllers - (a) linear PD, (b) typical fuzzy, (c) log-domain without a correction factor and (d) log-domain with a correction factor

4.1.3 Analysis

As can be seen from the surfaces of Figures 4.4(a) and 4.4(b), the typical fuzzy-logic controller developed in [78] is an approximation of the linear PD controller. For output values ranging from -150 to +150, the typical fuzzy controller is able to provide essentially the same results as the PD controller.

The proposed log-domain controller without correction factor makes use of only the maximum value to approximate the summation of a set of logarithmic values. While this approach relieves us from using computationally expensive multiplication and division operations, the RMS difference value between the typical and log-domain controller is higher (see also Figure 4.4(c)).

However, with a small correction factor involving the second highest value in a sequence, the approximate output becomes much more similar to that of typical fuzzy-logic controller. The corresponding RMS difference confirms the effectiveness of this approach through a greater than 50% reduction in value of

the difference. The impact of having a correction factor is also clear from the RMS difference value between the log-domain controllers. But when it comes to approximating the outputs with those of a PD controller, the correction factor does not seem to offer any advantage.

There is not much obvious difference among the step responses of the different controllers, as shown in Figure 4.3. Table 4.2, however, presents some very interesting insights. For each log-domain controller, the rise time is lower than both the linear PD and typical fuzzy controllers. Log-domain controllers also have lower settling times, although a small amount of overshoot is introduced.

4.2 Experiment 2

We have used three more second-order plants to verify and analyze the performance of our proposed controller. The plant transfer functions have the general form of $400/(s^2+ \sigma s)$, where σ is given three different values: 20, 48.5 and 360, respectively [77].

4.2.1 Controllers

The membership functions and rule base for the typical controller in [77] are shown in Figure 4.5 and Table 4.4, respectively. The value of (b_{i+1} - b_i) is equal to 1/18 for the first input (error), while (b_{i+1} - b_i) is 4000/18 for the second input (rate) with $b_0 = 0$. The centers of the output membership functions are defined as $[p_{-3}\ p_{-2}\ p_{-1}\ p_0\ p_1\ p_2\ p_3]$ = [-2500 -1000/3 -50/2 0 50/2 1000/3 2500].

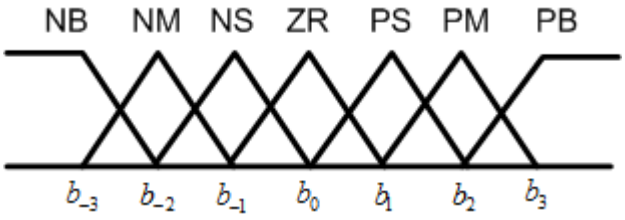


Figure 4.5 (a)

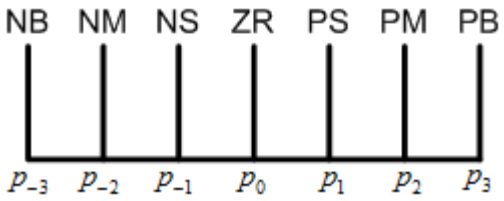


Figure 4.5 (b)

Figure 4.5: (a) Input and (b) output membership functions

Table 4.4: Rule base for the fuzzy controller [77]

Input2/Input1	NB	NM	NS	ZR	PS	PM	PB
PB	NB	NB	NB	NB	NM	NS	ZR
PM	NB	NB	NB	NM	NS	ZR	PS
PS	NB	NB	NM	NS	ZR	PS	PM
ZR	NB	NM	NS	ZR	PS	PM	PB
NS	NM	NS	ZR	PS	PM	PB	PB
NM	NS	ZR	PS	PM	PB	PB	PB
NB	ZR	PS	PM	PB	PB	PB	PB

Note that NB = Negative Big, NM = Negative Medium, NS = Negative Small, ZR = Zero, PS = Positive Small, PM = Positive Medium, and PB = Positive Big.

4.2.2 Results

Figures 4.6, 4.7 and 4.8 compare the step response of our log-domain controllers (both with and without correction factor) with a typical fuzzy controller for the plants mentioned above. Tables 4.5 to 4.7 present a comparison of rise time, settling time, and overshoot for different controllers and different plants.

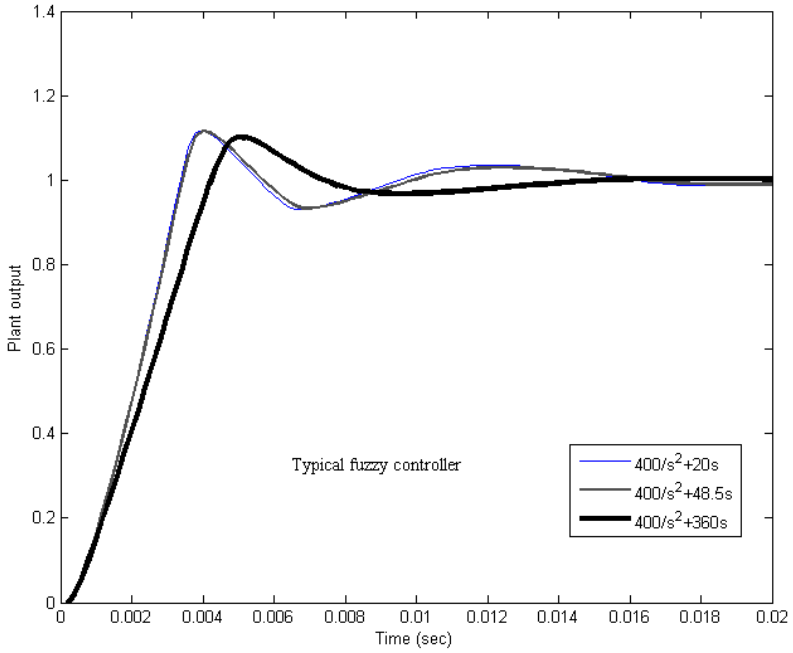


Figure 4.6: Step response of typical fuzzy controller

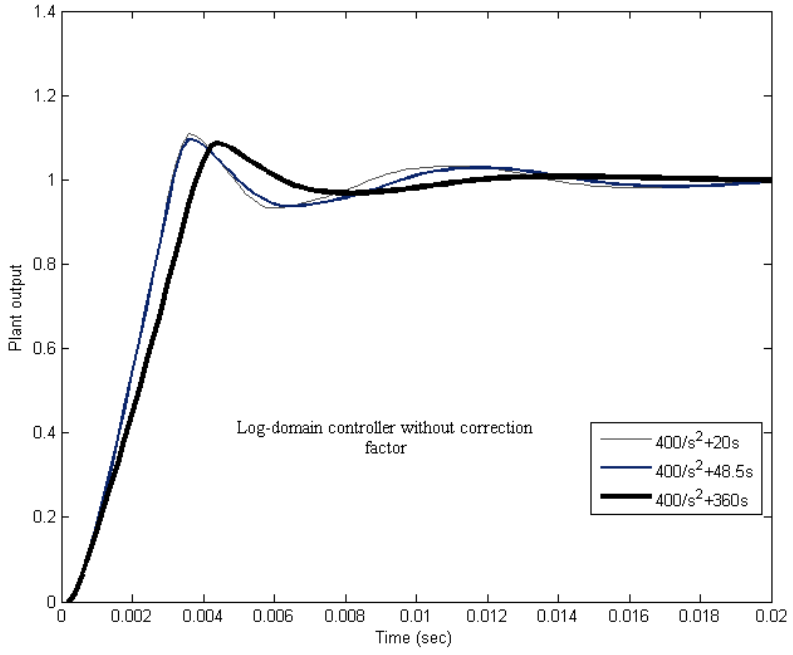


Figure 4.7: Step response of log-domain controller without correction factor

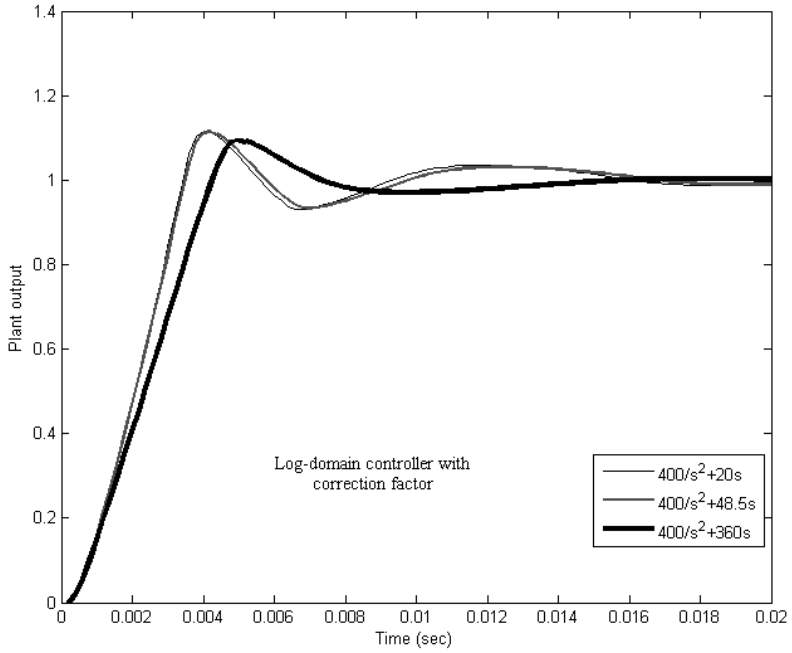


Figure 4.8: Step response of log-domain controller with correction factor

Table 4.5: Rise time, settling time and overshoot for the plant $400/(s^2+20s)$

Controllers	Rise Time (sec)	Settling Time (sec)	Overshoot (%)
Typical Fuzzy	0.0024	0.0145	11.7373
Log-domain (uncorrected)	0.0022	0.0126	10.8925
Log-domain (corrected)	0.0024	0.0145	11.5596

Table 4.6: Rise time, settling time and overshoot for the plant $400/(s^2+48.5s)$

Controllers	Rise Time (sec)	Settling Time (sec)	Overshoot (%)
Typical Fuzzy	0.0024	0.0145	11.5669
Log-domain (uncorrected)	0.0022	0.0132	9.6103
Log-domain (corrected)	0.0024	0.0148	11.4486

Table 4.7: Rise time, settling time and overshoot for the plant $400/(s^2+360s)$

Controllers	Rise Time (sec)	Settling Time (sec)	Overshoot (%)
Typical Fuzzy	0.0030	0.0122	10.1534
Log-domain (uncorrected)	0.0027	0.0099	8.7054
Log-domain (corrected)	0.0030	0.0123	9.4037

Control surface outputs generated for a typical fuzzy controller, and log-domain controllers are shown in Figures 4.9, 4.10 and 4.11. Table 4.8 illustrates root-mean-square (RMS) difference between control surfaces.

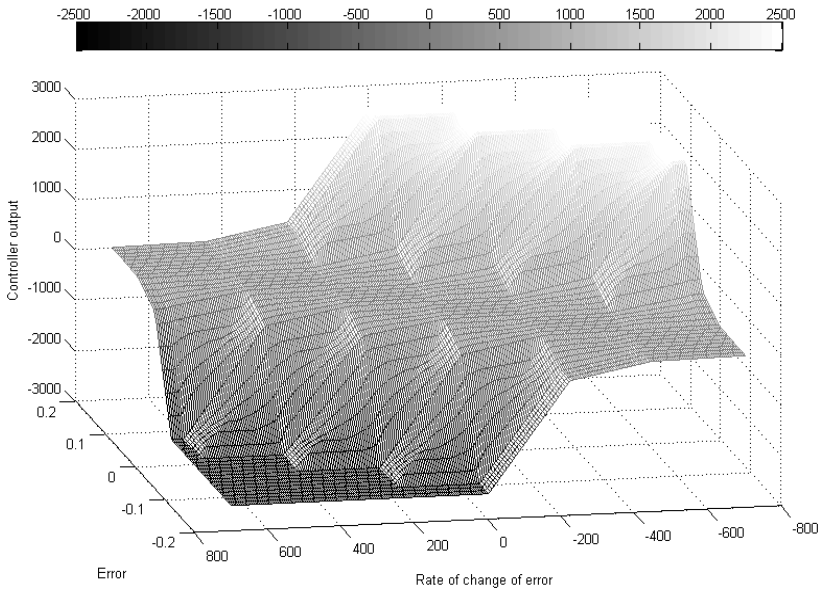


Figure 4.9: Control surface for typical fuzzy controller

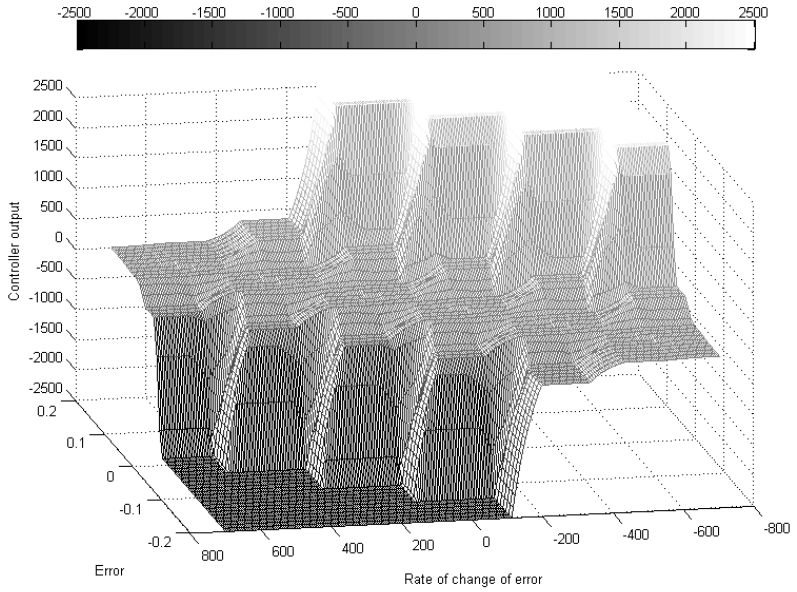


Figure 4.10: Control surface for log-domain controller without correction factor

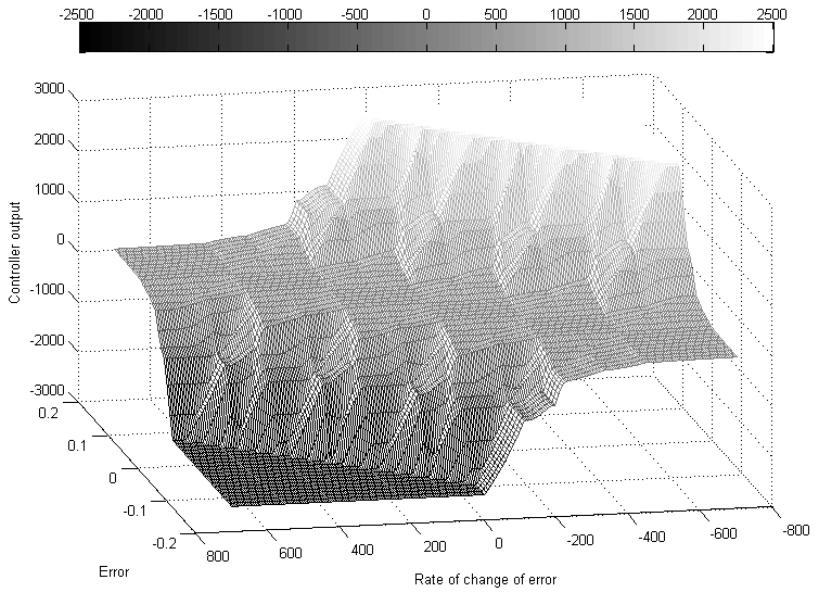


Figure 4.11: Control surface for log-domain controller with correction factor

Table 4.8: RMS difference between control surfaces

	Log-domain (uncorrected)	Log-domain (corrected)
Typical	355.0986	120.1319
Log-domain (uncorrected)		332.6803

4.2.3 Analysis

The step responses confirm the effectiveness of our proposed log-domain controllers. The plots from Figures 4.6, 4.7 and 4.8 are qualitatively similar, demonstrating that log-domain controllers perform as good as a typical fuzzy controller.

Tables 4.5, 4.6 and 4.7 present some interesting insights. The log-domain controller without correction factor performs better than both the typical fuzzy controller and its peer with correction factor for all cases of rise time, settling time and overshoot. The log-domain controller with correction factor has the same rise time and better overshoot compared to the typical controller for all the three plants, although the settling time is slightly higher for a couple of plants. This is a very promising result considering the fact that our log-domain controllers are much faster in processing than the typical fuzzy controller.

The control surface plots are also quite similar in shape for the log-domain and typical fuzzy controllers. Obviously, the surface of log-domain controller with correction factor resembles more with typical fuzzy controller, because of the approximate correction factor. Without this factor, the output is crudely approximated based on only the maximum value in a sequence. That is why there are some extra staircase outputs in the surface of log-domain controller without the factor. The RMS difference table also confirms this fact. Roughly a 67% reduction in output difference between the log-domain and typical fuzzy controller is obtained because of the correction factor. The impact of having a correction factor is also clear from the difference between the log-domain controllers.

CHAPTER 5

HARDWARE IMPLEMENTATION

5.1 Design Considerations

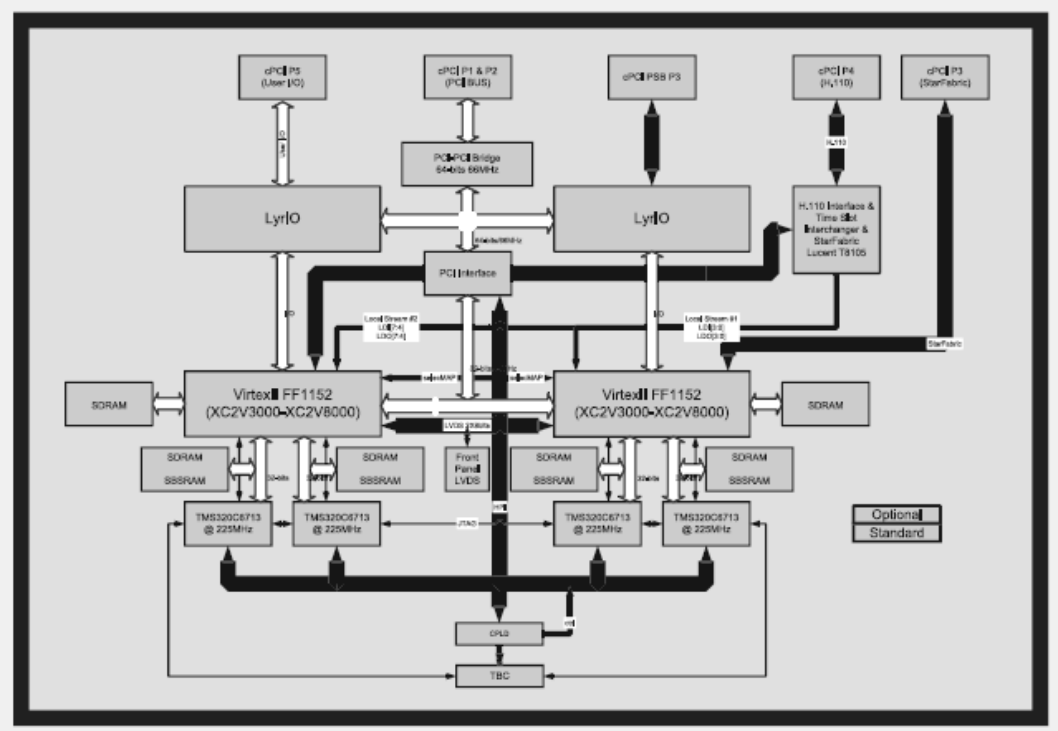


Figure 5.1: Block Diagram of Lyrtech SignalMaster board which consists of a DSP, a Virtex-II FPGA and multiple input/output expansions [49]

We have implemented the log-domain controller without the correction factor on a Xilinx Virtex – II FPGA (Figure 5.1) [45]. The plant and the rulebase are taken from [78]. The transfer function of the second-order plant is $\frac{1}{0.02s^2 + s}$. By using the substitution $s = \frac{z-1}{Tz}$, with T being the sampling period, as in the backward rule approach, the continuous transfer function is converted to the discrete transfer function of the form:

$$\frac{T^2}{(0.02+T)-(0.04+T)z^{-1}+0.02z^{-2}} \tag{5.1}$$

Since T = 0.01 sec, (5.1) becomes

$$\frac{0.0033}{1-1.667z^{-1}+0.667z^{-2}} \tag{5.2}$$

This corresponds to the standard form of second order digital filter:

$$\frac{a_0+a_1z^{-1}+a_2z^{-2}}{1+b_1z^{-1}+b_2z^{-2}} \tag{5.3}$$

Where a_0 = 0.0033, $a_1 = a_2$ = 0, b_1 = -1.667, and b_2 = 0.667.

This type of digital filter is implemented in the following form [76]:

$$y(k) = a_0x(k) + a_1x(k-1) + a_2x(k-2) - b_1y(k-1) - b_2y(k-2) \tag{5.4}$$

Where k = current sample in time, $x(k)$ = input to the plant at kth sample, and $y(k)$ = output from the plant at kth sample.

Figure 5.2 shows how (5.4) is implemented on FPGA [76].

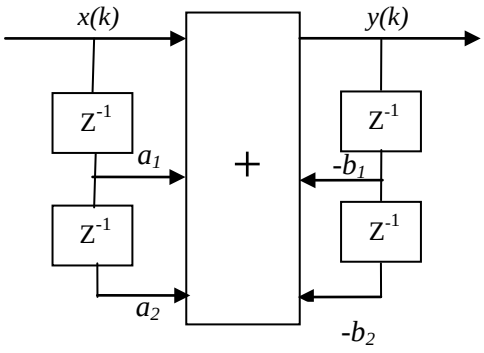


Figure 5.2: Second order digital filter implementation on FPGA

The fuzzifier module of the log-domain controller accepts two input values, namely – error and rate of error, both being 10-bit wide. The error value has 2 bits to represent the integer part and 8 bits for fractional part. The reason for using this

bit width is that simulation results have shown the possible error values ranging from 0.0 to 2.0. The rate values, however, can be anything between -39.0 and 0.0. Therefore, 10 bits for rate input actually represent 6 bits for integer and 4 bits for fractional part. As the rate values are negative, we are only considering the absolute quantities without the sign. The lower number of bits for fractional part does not have any visible impact because of the rate values being sparse. This bit width also determines the size of the lookup tables (LUTs) being used in the fuzzifier module. The error and rate LUTs have 1024 rows (all combinations of 10 bits) and five columns representing the logarithm of membership function values for NB, NM, ZR, PM and PB. The log values stored in the LUTs are all negative; therefore the sign is ignored. Each value is 16-bit wide with 4 bits for integer part and 12 bits for fractional part. 4 bits can represent as high as 15 which is enough for this context, since the maximum log value can be 9.0. Another LUT contains 4096 rows each having a possible exponential value for an output value. Since the logarithm of the output can take any value within the range -11.0 to +5.0, we are using 1 sign bit, 3 bits for integer part and 8 bits for fractional part. The use of 3 integer bits can only represent a negative number as low as -7; but it is justified by the fact that any lower number results in the exponential value being essentially zero. The output from the defuzzifier is represented by 20 bits, where 1 bit is used as sign, 7 bits for integer part and 12 bits for fractional part. Since the fuzzy controller output can go as high as 100.0, we have to use at least 7 bits to represent the value. Note that for a generalized hardware log-domain controller, the bit widths may be different making the rows and columns in the LUTs bigger or smaller. The performance, however, should be the same except for a different amount of resource utilization.

The step signal is implemented on the same FPGA as the plant model and the fuzzy inference engine as a square wave to enable the plant outputs to be shown on the oscilloscope. On the positive cycle, the step value goes to 2 from 0; therefore, the first error value becomes 2.0 while the first rate value is zero. These values are sent to the log-domain controller which produces a defuzzified output to be passed to the plant. The plant output is compared to 2.0 and the difference

becomes the next error value. The next rate value is calculated based on the current and the previous error values. Eventually both the error and rate values become zero.

5.2 Log-domain Controller

The block diagram of our log-domain controller with the plant implemented on FPGA is shown in Figure 5.3. Note that, unlike the plant (expressed in transfer function) in the simulation; here the plant is implemented as a second-order digital filter as described in Section 5.1.

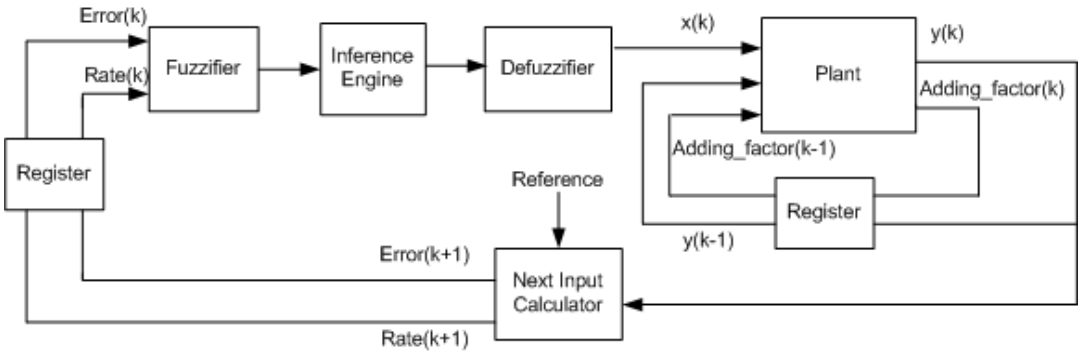


Figure 5.3: Block diagram of log-domain controller with plant as implemented on Xilinx Virtex-II FPGA

The fuzzifier module passes the corresponding row from both of the error and rate LUTs to the inference engine. The inference engine calculates the maximum value (log of firing strength for a rule) for each of the 25 possible pairs. Although the conventional task of inference engine is to calculate the minimum values, we are using maximum instead, since all the negative log values are stored as positive. The six maximum values are passed to the defuzzifier module.

The defuzzifier module subtracts the log of firing strength values (log_fs) from the log of consequent values (log_c), and stores the results in an array named log_d. The maximum value among log_d ($\log\_d_{max}$) is calculated, while the minimum for log_fs is also determined. These two values are added together and

the summation result (log_output) is used to find the corresponding exponential value from a LUT. This exponential value is assigned to the output, when the particular value of log_c (that contributes to the calculation of the value log_d_{max}) comes from a positive consequent value. Otherwise the exponential value is made negative before assigning it to the final output.

The output from the log-domain controller goes to the plant. It also accepts the plant output and an adding factor from the previous sample. The previous adding factor is used to calculate the current plant output, whereas the previous plant output contributes to generate the new adding factor. The fuzzy controller output (*x(k)*) gets multiplied by a_0 (0.0033) and added to Adding_factor(k-1) to generate the new plant output (*y(k)*). Adding_factor(k) is calculated as

$$Adding\_factor(k) = -b_{1*}y(k) - b_{2*}y(k-1)$$

The plant output (*y(k)*) is passed to Next Input Calculator which generates the next error and rate values. Error(k+1) is calculated as step minus plant output at k^{th} sample, whereas Rate(k+1) is calculated in two steps. First, new error value is subtracted from the previous error value, and then the result is multiplied by 1/T (100.0). It is worth mentioning that all the multiplication operations are basically a group of additions because one of the operands is always a constant value.

Figure 5.4 illustrates the operations that take place in the control system.

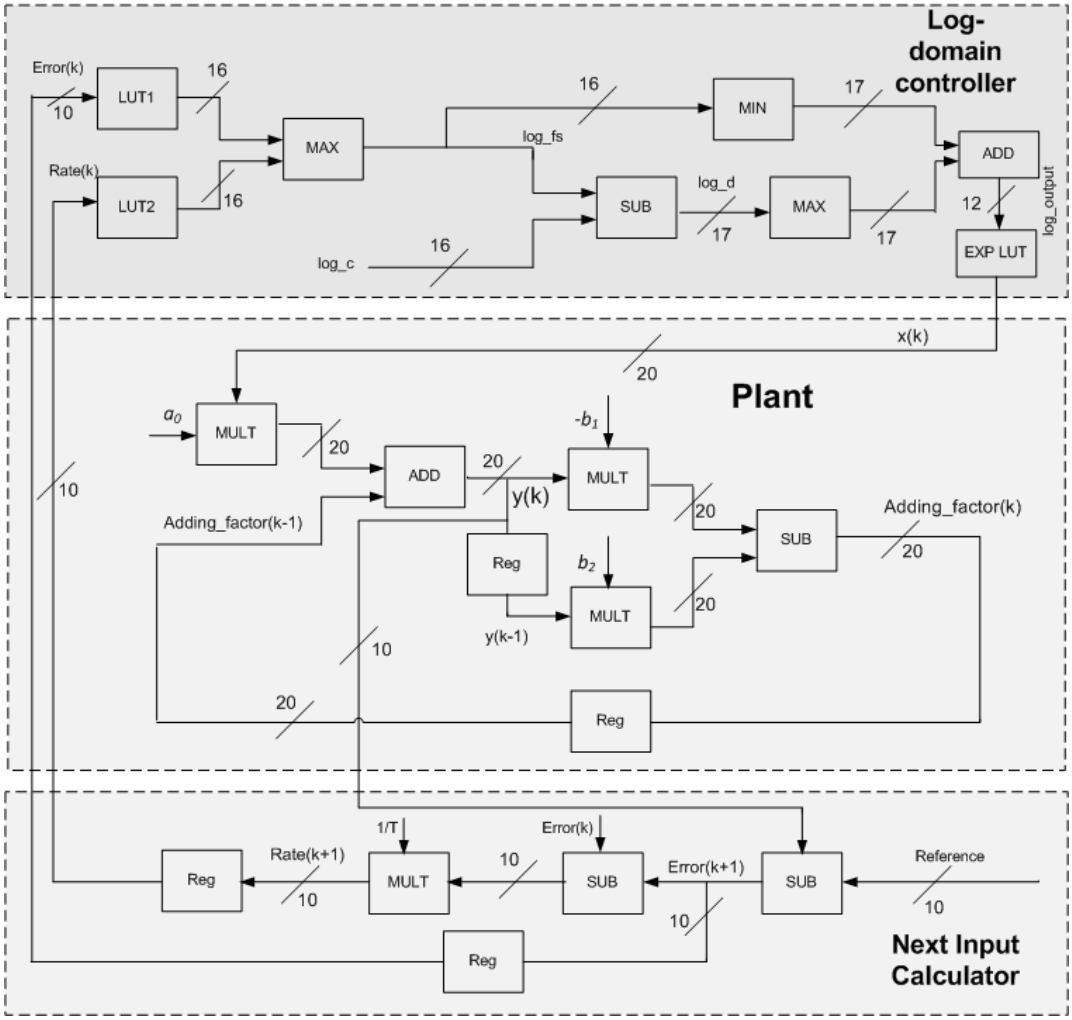


Figure 5.4: Operations inside the control system

5.2.1 Results

Figure 5.5 displays the plant outputs on an oscilloscope for the log-domain controller. Here the positive cycle refers to the plant outputs for a step signal. The negative cycle simply mirrors the plant outputs along the horizontal axis. Note that in Figure 5.5, one waveform represents the reference (Figure 2.1) and the other waveform represents the step response, i.e. the outputs of the plant.

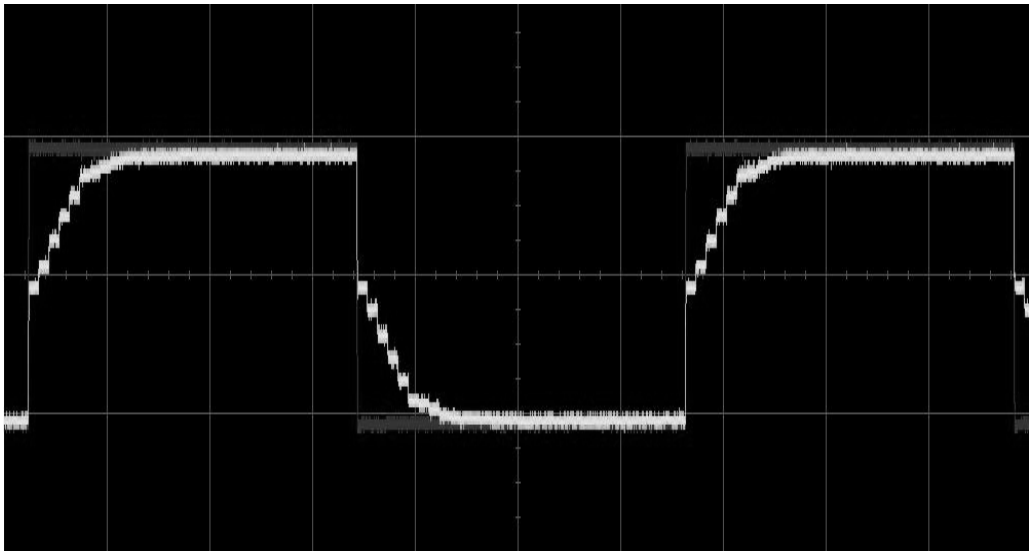


Figure 5.5: Plant outputs for log-domain controller implemented on FPGA and displayed on an oscilloscope for both positive and negative cycles

Rise Time

In our implementation, rise time means the time for the plant output to reach a value of 1.8 from 0.2. Since it takes 8 samples to get to 1.8 and each sample takes 14.8 ns, so rise time is (8*14.8)ns = 118.4 ns. In the simulation results, rise time was 0.0567 sec with a sampling period of 0.01 sec. This means, about 6 samples were needed in simulation.

Settling Time

Settling time is the time when the plant output reaches a value of at least 1.95 and never goes lower. As 11 samples are required to settle down for the plant outputs in our implementation, settling time is (11*14.8)ns = 162.8 ns. Settling time was shown to be 0.1 sec in the simulation results, which correspond to 10 samples.

Overshoot

Overshoot occurs when the plant output goes beyond the steady-state value, which, for our log-domain controller, does not happen. So overshoot is zero. For the simulation experiments, overshoot was 0.0009%.

Speed

From the synthesis report, the maximum clock frequency of our log-domain controller is 67.6 MHz; since our controller processes one plant output per clock cycles, this implies a processing speed of 67.6 MFLIPS. This is an improvement over the reported fastest fuzzy controller implementation [80] on 1.0 μm CMOS technology having a speed of 50 MFLIPS. It demonstrates the effectiveness of our approach in using the log-domain arithmetic.

Resource Utilization

Table 5.1 shows the resource utilization for our controller on FPGA – one with the plant model and the other one without the plant. It can be seen from the table that the log-domain controller uses a very small percentage of the overall available resources.

Table 5.1: FPGA resource utilization for log-domain controller

Resources	**Log--domain controller with plant**	**Log--domain controller without plant**
ROMS	3	3
Adders/Subtractors	1	1
Counters	2	2
Registers	84	84
Comparators	9	9
Priority Encoders	6	6
XORs	1363	243
Slices	118 out of 46592	65 out of 46592
Slice Flip-flops	62 out of 93184	53 out of 93184
Bonded IOBs	201 out of 824	201 out of 824

5.3 Pipelined Log-domain Controller

We have already shown a clear speedup for our controller compared to the 50 MFLIPS implementation [80]. One drawback of this type of feedback system is that the inputs at any sample depend on the output from previous sample. So the system cannot accept inputs at every clock cycle; rather it waits several clock cycles so that the plant output for a particular input set gets calculated. This results in a substantial waste of resources for systems where there is no dependency among the inputs. We believe our proposed controller is more suitable for networking applications (e.g. per-packet inspection) where inputs would be independent of one another and the output ports cannot be idle for the entire processing time. That is why we have designed a pipelined version of our controller which generates a plant output at every clock cycle.

As four clock cycles are required to generate a plant output after the error and rate inputs are applied, we use pre-calculated error and rate inputs for the first four cycles so that plant outputs start to be generated from fifth cycle onward. When a new pair of error and rate inputs (*Error(k)* and *Rate(k)*) is applied to the fuzzifier, the membership values stored in the LUTs are calculated instantaneously and passed to the inference engine without any delay. At the same time, the error value and plant output at the current cycle is passed to Next Input Calculator module to calculate the next error and rate values. The inference engine implements the maximum function for each combination of membership values, and the result is passed to the defuzzifier at the start of next clock cycle (pipeline stage 1).

The defuzzifier first calculates *log_d* values and then finds the maximum among them. At the same time, the minimum value among the log of firing strength values is determined. Both these values are sent for addition at the next cycle (pipeline stage 2). The summation result is used for an exponential value from a LUT and sign is adjusted before it gets to the plant at the next cycle (pipeline stage 3). Fuzzy controller output (*x(k)*) is multiplied by a_0 and the result is added to *Adding_factor(k-1)* to calculate the plant output (*y(k)*) at the next cycle

(pipeline stage 4). The plant output is multiplied by ($-b_1$), while previous plant output is multiplied by b_2. They are passed to a subtractor block to calculate the adding factor at the following clock cycle. Although *Adding_factor(k)* is generated one cycle later than the plant output; it does not make any difference in the overall system performance, because it is used only after the next plant output is calculated, which takes place 3 cycles later.

So at the start of 5th cycle we have a plant output for the inputs applied in the 1st cycle. The new pair of input values produced in the first cycle is sent to the 2nd cycle to generate a plant output at the 6th cycle. This continues and eventually the plant outputs become stable.

Figure 5.6 shows the block diagram of the pipelined version of the log-domain controller. REG blocks refer to the registers being used to accomplish the pipelining stages.

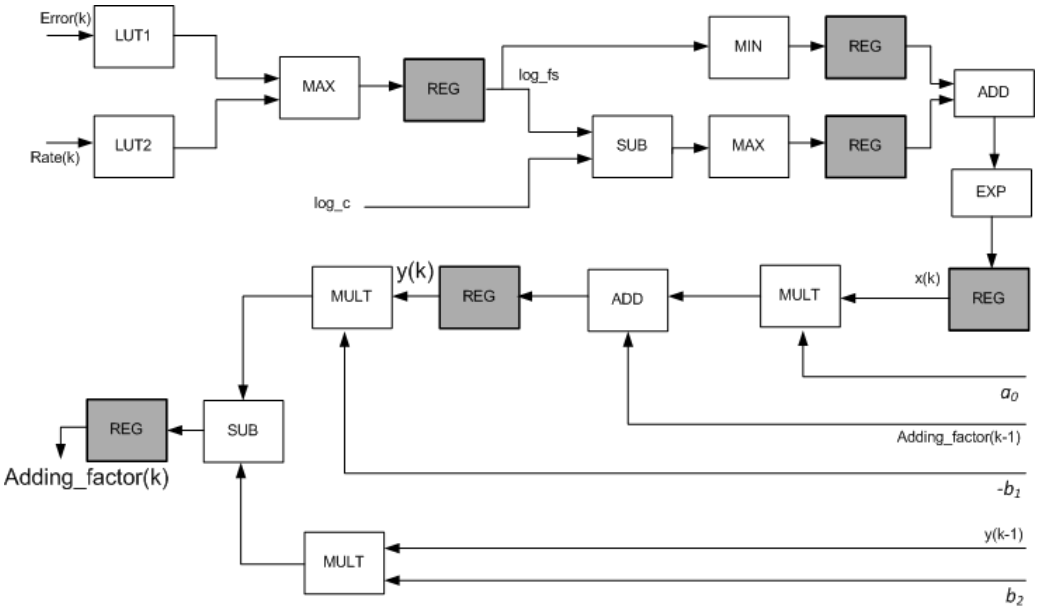


Figure 5.6: Pipeline stages of log-domain controller

5.3.1 Results

Figure 5.7 shows the oscilloscope plot for pipelined version of the controller.

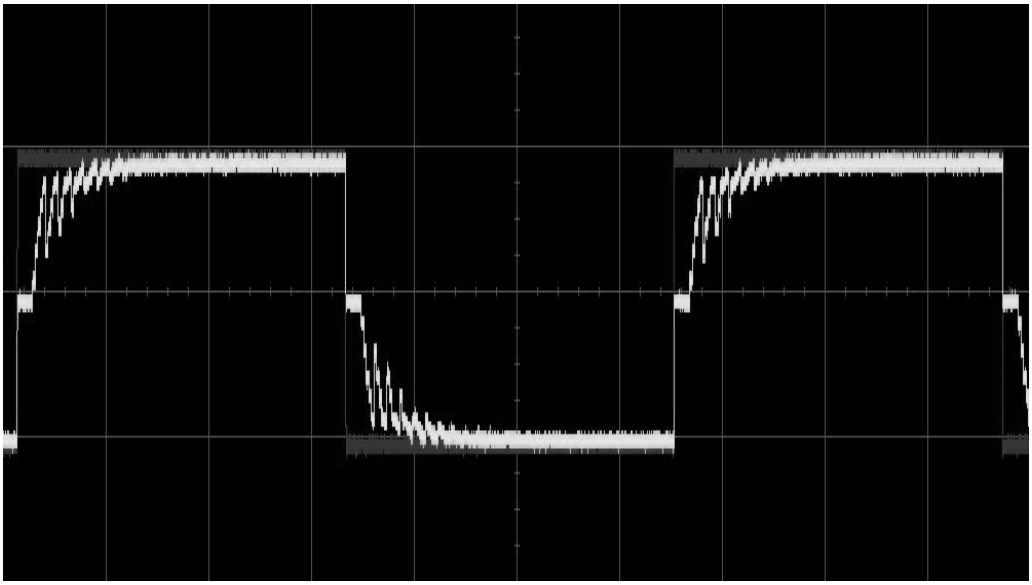


Figure 5.7: Plant outputs for pipelined log-domain controller implemented on FPGA and displayed on an oscilloscope for both positive and negative cycles

Rise Time

Since it takes 27 cycles for plant outputs to reach to 1.8, rise time is (27*4.02) ns = 108.5 ns.

Settling Time

46 cycles are required to reach within 2.5% of the step signal, so settling time is (46*4.02) ns = 184.9 ns.

Overshoot

Overshoot is zero, because plant output never exceeds the step signal.

Speed

The pipelining results in a processing speed of 248.7 MFLIPS, a significant improvement in fuzzy controller implementation in hardware.

Resource Utilization

Table 5.2 illustrates the resource utilization on FPGA for our pipelined log-domain controller – one with the plant model and another without the plant. Similar to its non-pipelined version, the pipelined controller still uses a limited number of resources; although resource consumption is slightly higher in several cases. Less registers and XORs are used when we do not consider the plant on FPGA.

Table 5.2: FPGA resource utilization for pipelined log-domain controller

Resources	**Pipelined log-domain controller with plant**	**Pipelined log-domain controller without plant**
ROMS	4	4
Adders/Subtractors	1	1
Counters	2	2
Registers	642	572
Comparators	9	9
Priority Encoders	6	6
XORs	1363	243
Slices	196 out of 46592	196 out of 46592
Slice Flip-flops	257 out of 93184	257 out of 93184
Bonded IOBs	201 out of 824	201 out of 824

5.4 Comparative Analysis

Comparisons of rise time, settling time, overshoot and speed between non-pipelined and pipelined version of log-domain controller are shown in Table 5.3.

Table 5.3: Comparisons of rise time, settling time, overshoot and speed between non-pipelined and pipelined version of log-domain controller

	Log-domain controller (without pipelines)	Log-domain controller (with pipelines)
Rise Time (ns)	118.4	108.5
Settling Time (ns)	162.8	184.9
Overshoot (%)	0	0
Speed (MFLIPS)	67.6	248.7

As can be seen from Table 5.3, and Figures 5.5 and 5.7, the pipelined version has lower rise time, but higher settling time. The longer settling time results from the initial oscillation in the plot for pipelined controller. The oscillation is due to the dependency of the plant outputs, which is natural for feedback systems. However, for real-life networking applications, where each input set will be independent of one another, our log-domain controller is capable of achieving very fast processing speed as demonstrated by the speed comparisons between two versions of the controller.

In terms of the resource utilization, non-pipelined version uses fewer number of ROMs, registers, slices and slice flip-flops, because of the fact that pipelining has some processing overhead. This is compensated by a speedup of almost 4 times due to pipelining.

CHAPTER 6
FUTURE RESEARCH

The log-domain fuzzy controller confirms the effectiveness of the proposed approach for a number of second-order plants in the simulation of step responses and control surfaces. The hardware versions achieve a significant speedup compared to the existing fuzzy controllers. Our future research will focus on implementing this log-domain technique for more higher-order plants to gain further insights into this method. One other goal will be to develop a computationally simpler correction factor to compensate for the performance penalty. Finally, although we have limited our study to FPGA-based implementations, we feel that a valid direction of future research would investigate the performance of fuzzy inference engines on current state-of-the-art microprocessors, including the application of our log-domain techniques.

CHAPTER 7
CONCLUSION

In this thesis, a logarithmic arithmetic based fuzzy logic controller is described, which results in a very high-speed hardware implementation. This approach gets rid of computationally expensive multiplication and division operations that have been the bottlenecks for fuzzy control systems. As the simulation results illustrate, log-domain implementations – with and without the approximate correction factor – perform better than the typical fuzzy and linear PD controllers in terms of rise time and settling time of the step response curves. Although there has been a small approximation error in the control surfaces, this does not have much impact on the system performance as depicted by the step response. The hardware implementation on FPGA without the correction factor also achieves similar response to the step signal. The processing speed of the hardware version is 67.6 MFLIPS which exceeds the fastest fuzzy controller implementation in literature that we are aware of by 33%. Note, however, that the implementation [80] was performed in 1.0 μm CMOS VLSI technology in 1995 with four 7-bit inputs, one 7-bit output, 7 membership functions for each variable, and up to 127 rules. If we had implemented this controller on the same Xilinx Virtex-II FPGA that we have used for our log-domain controller, it would have resulted in a different speed. Nevertheless in this thesis, we have been able to develop a very fast fuzzy controller using logarithmic arithmetic, which is a totally new approach to design fuzzy logic control systems. The experiments have shown the ability of the controllers to produce outputs that are close to the expected ones. A further speedup to 248.7 MFLIPS is also achieved by a pipelined version of the log-domain controller. We believe that this is a very promising result making the log-domain controllers potentially suitable for high-speed and large networks.

BIBLIOGRAPHY

[1] Levine, W. S. (1996). *The Control Handbook*. New York: CRC Press.

[2] Macia, N. F., & Thaler, G. J. (2004). *Modeling and Control of Dynamic Systems*. Albany: Delmar Cengage Learning.

[3] Wade, H. L. (2004). *Basic and Advanced Regulatory Control: System Design and Application*. Isa-The Instrumentation, Systems, and Automation Society.

[4] Goodwin, G. C., Graebe, S. F., & Salgado, M. E. (2001). *Control System Design*. Alexandria, VA: Prentice Hall.

[5] J.D-Azzo, J., Houpis, C. H., & Sheldon, S. N. (2003). *Linear Control System Analysis and Design with Matlab, Fifth Edition*. CRC Press.

[6] Nagle, C. L., & Phillips, H. T. (1995). *Digital Control System Analysis and Design*. Alexandria, VA: Prentice Hall.

[7] Pedrycz, W., & Gomide, F. (2007). *Fuzzy Systems Engineering*. Hoboken, New Jersey: John Wiley & Sons Inc.

[8] Jantzen, J. (2007). *Foundations of Fuzzy Control*. New York, NY: Wiley.

[9] Zadeh, L. (1965). Fuzzy Sets. *Information and Control*, *8*(*3*), 338-353.

[10] Zadeh, L. (1968). Fuzzy Algorithms. *Information and Control*, 12, 94-102.

[11] Mamdani, E. H. (1977). Application of Fuzzy Logic to Approximate Reasoning. *IEEE Trans. Computers*, Issue 26, 1182-1191.

[12] Mamdani, E. H. (1974). Application of Fuzzy Algorithms for Control of a Simple Dynamic Plant. *Proc. IEEE*, Issue 121, 1585-1588.

[13] Stoll, R. R. (1979). *Set Theory and Logic*. New York: Dover Publications.

[14] Filev, D. P., & Yager, R. R. (1994). *Essentials of Fuzzy Modeling and Control*. New York, NY: Wiley.

[15] Jang, J. R., Mizutani, E., & Sun, C. (1996). *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Alexandria, VA: Prentice Hall.

[16] Nguyen, H. T., & Walker, E. A. (2000). *A First Course in Fuzzy Logic*. New York: Chapman & Hall.

[17] Gerla, G. (2005). *Fuzzy Logic Programming and Fuzzy Control*, Studia Logica, 79, 231-254.

[18] Ibrahim, A. (2004). *Fuzzy Logic for Embedded Systems Applications*. Elsevier.

[19] Kim, Y. D. (1997). High Speed Flexible Fuzzy Hardware for Fuzzy Information Processing. *IEEE Trans. on Systems, Man, and Cybernetics – Part A,* 27 (1), 45-56.

[20] Lee, C. S., & Lin, C. (1996). *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Alexandria, VA: Prentice Hall.

[21] Klir, G. J., & Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Upper Saddle River: Prentice Hall PTR.

[22] Lee, S. G., & Carpinalli, J. D. (2005). High-speed Integer Operations in the Fuzzy Consequent Part and the Defuzzification Stage for Intelligent Systems. *International Journal of Intelligent Control and Systems.* 10 (4), 258-268.

[23] Manaresi, N., Rovatti, R., Franchi, E., Guerrieri, R., & Baccarani, G. (1996). Automatic synthesis of analog fuzzy controllers: a hardware and software approach. *IEEE Transactions on Industrial Electronics*, *43*(*1*), 217 - 225.

[24] Dick, S., Gaudet, V., & Bai, H. (2008). Bit-serial arithmetic: A novel approach to fuzzy hardware implementation. *Annual Meeting of the North American Fuzzy Information Processing Society*, 1-6.

[25] Kalaykov, I. & Tolt, G. (2002). Fast fuzzy signal and image processing hardware. *Annual Meeting of the North American Fuzzy Information Processing Society*, 7-12.

[26] Yamakawa, T., & Miki, T. (1986). The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process . *IEEE Transactions on Computers, C-35*(*2*), 161-167.

[27] Chen, J. -J., Chen, C. -C., & Tsao, H. -W. (1992). Turnable membership function circuit for fuzzy control systems using CMOS technology. *Electronics Letters*, *28*(*22*), 2101-2103.

[28] Sasaki, M., Ishikawa, N., Ueno, F., & Inoue, T. (1992). Current-mode analog fuzzy hardware with voltage input interface and normalization locked loop. *IEEE International Conference on Fuzzy Systems*, 451-457.

[29] Yamakawa, T. (1993). A fuzzy inference engine in nonlinear analog mode and its application to a fuzzy logic control. *IEEE Transactions on Neural Networks*, *4*(*3*), 496-522.

[30] Ota, Y., & Wilamowski, B. (1996). CMOS Implementation of a Voltage-Mode Fuzzy Min-Max Controller. *Journal of Circuits, Systems, and Computers*, *6*(*2*), 171-184.

[31] Control System Stability. (n.d.). *RoyMech Index page*. Retrieved December 30, 2009, from http://www.roymech.co.uk/Related/Control/Stability.html.

[32] Baturone, I., Huertas, J., Barriga, A., & Sánchez, S. (1994). Current-mode multiple-input Max circuit. *Electronics Letters*, *30*(*9*), 678-680.

[33] Introduction to Systems Engineering. (n.d.). *Federation of American Scientists*. Retrieved December 30, 2009, from http://www.fas.org/man/dod-101/navy/docs/es310/Int_SysE/Int_SysE.htm.

[34] Liu, B., Huang, C., & Wu, H. (1994). Modular current-mode defuzzification circuit for fuzzy logic controllers. *Electronics Letters*, *30*(*16*), 1287-1288.

[35] Baturone, M., Sánchez-Solano, S., & Huertas, J. (1994). Current-mode singleton fuzzy controller. *3rd. International Conference on Fuzzy Logic, Neural Nets and Soft Computing (IIZUKA'94)*, 647-648.

[36] Tham, M. (n.d.). Mathematics of Sampled Data Systems. *Newcasle University*. Retrieved December 30, 2009, from http://lorien.ncl.ac.uk/ming/digicont/digimath/sampled2.htm.

[37] Smith, J. (n.d.). Frequency Warping. *Introduction to Digital Filters with Audio Applications*. Retrieved December 29, 2009, from http://ccrma.stanford.edu/~jos/filters.

[38] Ken, J., Li, W., & Liu, J. (2008). Fuzzy Immune Self-tuning PID Controller and Its Simulation. 3rd IEEE Conference on Industrial Electronics and Applications, 625 - 628.

[39] Nagle, R. K., & Saff, E. B. (1993). *Fundamentals of Differential Equations* (3 ed.). Toronto: Addison Wesley.

[40] Bouras, S., Kotronakis, M., Suyama, K., & Tsividis, Y. (1998). Mixed analog-digital fuzzy logic controller with continuous-amplitude fuzzy inferences and defuzzification. *IEEE Transactions on Fuzzy Systems*, *6*(*2*), 205-215.

[41] Braun, M. (1992). *Differential Equations and Their Applications: An Introduction to Applied Mathematics* (4th ed.). New York: Springer.

[42] Dukkipati, R. V. (2005). *Control Systems*. London: Alpha Science International, Ltd.

[43] Pierzchala, E., Perkowski, M., & Grygiel, S. (1994). A field programmable analog array for continuous, fuzzy, and multi-valued logic applications. *Twenty-Fourth International Symposium on Multiple-Valued Logic*, 148-155.

[44] Manaresi, N., Franchi, E., Guerrieri, R., & Baccarani, G. (1996). A field programmable analog fuzzy processor with enhanced temperature performance. *Proc. 22nd European Solid-State Circuits Conf. (ESSCIRC‘ 96)*, 152–155.

[45] SignalMaster Quad. (n.d.). *Lyrtech DSP-FPGA design house, digital signal processing product design services*. Retrieved December 14, 2009, from http://www.lyrtech.com/index.php?act=view&pv=SignalMaster%20Quad.

[46] Miki, T., & Yamakawa, T. (1995). Fuzzy inference on an analog fuzzy chip. *IEEE Micro*, *15*(*4*), 8-18.

[47] Eichfeld, H., Kunemund, T., & Menke, M. (1996). A 12b general-purpose fuzzy logic controller chip. *IEEE Transactions on Fuzzy Systems*, *4*(*4*), 460-475.

[48] Baturone, I., Barriga, A., Sánchez-Solano, S., & Huertas, J. (1998). Mixed-signal design of a fully parallel fuzzy processor. *Electronics Letters*, *34*(*5*), 437-438.

[49] Lyrtech Inc. (n.d.). *Quad TMS320C6713/ Dual Virtex-II-based SignalMaster*. Retrieved January 26, 2010, from http://www.dspecialists.de/pix/inhalt/lyrtech/pdf/1_3_signalmaster_quad_tms320c6713_en.pdf.

[50] Tigaeru, L. (2003). Programmable analogue membership function circuit for hybrid-mode fuzzy systems. *Electronics Letters*, *39*(*8*), 642–644.

[51] Han, I. S. (2007), Membership Function Circuit for Neural/Fuzzy Hardware of Analog-Mixed Operation Based on the Programmable Conductance, *IEEE Int. Fuzzy Systems Conf.*, 1-4.

[52] Guo, S., Peters, L., & Surmann, H. (1996). Design and application of an analog fuzzy logic controller. *IEEE Transactions on Fuzzy Systems*, *4*(*4*), 429 - 438.

[53] Baturone, I., Sanchez-Solano, S., Barriga, A., & Huertas, J. (1997). Implementation of CMOS fuzzy controllers as mixed-signal integrated circuits. *IEEE Transactions on Fuzzy Systems*, *5*(*1*), 1-19.

[54] Kim, Y., & Lee-Kwang, H. (1997). High Speed Flexible Fuzzy Hardware for Fuzzy Information Processing. *IEEE Transactions On Systems, Man, and Cybernetics*, *27*(*1*), 45-56.

[55] Saito, Y. (1989). A high speed software fuzzy inference controller. *Proceedings of the 3rd IFSA World Congress*, 12-15.

[56] Eshera, M., & Barash, S. (1989). Parallel rule-based fuzzy inference on mesh-connected systolic arrays. *IEEE Expert*, *4*(*4*), 27-35.

[57] Tombs, J., Torralba, A., & Franquelo, L. (1994). Design of a Fuzzy Controller Mixing Analog and Digital Techniques. *Proceedings of the Third IEEE Conference on Fuzzy Systems*, 1755 - 1758.

[58] Hung, D. (1995). Dedicated digital fuzzy hardware. *IEEE Micro*, *15*(*4*), 31-39.

[59] Togai, M., & Watanabe, H. (1986). Expert System on a Chip: An Engine for Real-Time Approximate Reasoning. *IEEE Expert*, *1*(*3*), 55-62.

[60] Togai, M., & Watanabe, H. (1986). VLSI implementation of a fuzzy-inference engine: Toward an expert system on a chip. *Information Sciences*, *38*, 147-163.

[61] Watanabe, H., Symon, J., Dettloff, W., & Yount, K. (1991). VLSI fuzzy chip and inference accelerator board systems. *Proceedings of the Twenty-First International Multiple-Valued Logic*, 120-127.

[62] Eichfeld, H., Lohner, M., & Muller, N. (1992). Architecture of a CMOS fuzzy logic controller with optimized memoryorganisation and operator design. *IEEE International Conference on Fuzzy Systems*, 1317-1323.

[63] Chiueh, T. (1992). Optimization of fuzzy logic inference architecture. *Computer, 25*(*5*), 67-71.

[64] Watanabe, H. (1992). RISC approach to design of fuzzy processor architecture. *IEEE International Conference on Fuzzy Systems*, 431-441.

[65] Hung, D. (1994). Custom design of a hardware fuzzy logic controller. *Proceedings of the Third IEEE Conference on Fuzzy Systems*, 1781-1785.

[66] Hung, D., & Zajak, W. (1995). Design and implementation of a hardware fuzzy inference system. *Information Sciences-Applications, 3*(*3*), 193-207.

[67] Salvador, L. D., & Gutierrez, J. (1995). A multilevel systolic approach for fuzzy inference hardware. *IEEE Micro, 15*(*5*), 61-71.

[68] Costa, A., Gloria, A. D., & Olivieri, M. (1996). Hardware design of asynchronous fuzzy controllers. *IEEE Transactions on Fuzzy Systems, 4*(*3*), 328-338.

[69] Kim, Y., Park, K., & Leekwang, H. (1995). Parallel fuzzy information processing system. *Fuzzy Sets and Systems, 72*(*3*), 323-329.

[70] Sánchez-Solano, S., Barriga, A., Jiménez, C., & Huertas, J. (1997). Design and application of digital fuzzy controllers. *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems*, 869-874.

[71] Ikeda, H., Kisu, N., Hiramoto, Y., & Nakamura, S. (1992). A fuzzy inference coprocessor using a flexible active-rule-driven architecture. *IEEE International Conference on Fuzzy Systems*, 537-544.

[72] Tokunaga, H., Terano, T., Terano, M., Mukaidono, M., & Shigemasu, K. (1992). Fuzzy computer prototype system—FUTURE BOARD system. *Fuzzy Engineering Toward Human Friendly Systems*, 1106–1107.

[73] Ghosh, S., Razouqi, Q., Schumacher, H., & Celmins, A. (1998). A survey of recent advances in fuzzy logic in telecommunications networks and new challenges. *IEEE Transactions on Fuzzy Systems*, *6*(*3*), 443-447.

[74] Lee, S., & Bien, Z. (1994). Design of expandable fuzzy inference processor. *IEEE Transactions on Consumer Electronics*, *40*(*2*), 171-175.

[75] Evmorfopoulos, N., & Avaritsiotis, J. (1999). Adaptive digital fuzzy hardware in application-specific integrated circuits. *The 6th IEEE International Conference on Electronics, Circuits and Systems*, 1635-1638.

[76] Gwaltney, D. King, K., Smith, K. & Ormsby, J. (2002), Implementation of Adaptive Digital Controllers on Programmable Logic Devices, https://www.researchgate.net/publication/23892297_Implementation_of_Adaptive_Digital_Controllers_on_Programmable_Logic_Devices.

[77] Taur, J. & Tao, C. (1997). Design and Analysis of Region-Wise Linear Fuzzy Controllers. *IEEE Trans. On Systems, Man and Cybernetics*, 27(3), 526-532.

[78] Kawaji, S., Maeda, T. & Matsunaga, N. (1991). Fuzzy Control Using Knowledge Acquired from PD Control, Industrial Electronics, *Control and Instrumentation*, vol 2, 1549-1554.

[79] Ascia, G., Catania, V., & Russo, M. (1999). VLSI hardware architecture for complex fuzzy systems. *IEEE Transactions on Fuzzy Systems*, *7*(*5*), 553-570.

[80] Gabrielli, A., Gandolfi, E., Masetti, M., & Russo, M. (1995). Design of a VLSI very high speed reconfigurable digital fuzzy processor. *Proc. ACM Symp. Appl. Comput.*, 477–481.

[81] Gross, W. & Gulak, G. (1998). Simplified MAP Algorithm Suitable for Implementation of Turbo Decoders. *IEE Electronics Letters*, 34 (16), 1577-1578.

[82] Erfanian, J., Pasupathy, S., & Gulak, G. (1990). Reduced Complexity Symbol Detectors with Parallel Structures. *IEEE GLOBECOM*, 704-708.

[83] Robertson, P., Villebrun, E., & Hoeher, P. (1995). A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain. *IEEE Int. Conf. Communications*, Seattle, WA, 1009-1013.

[84] Costa, A., Gloria, A. D., Faraboschi, P., & Pagni, A. (1995). Hardware solutions for fuzzy control. *Proceedings of the IEEE, 83*(*3*), 422-434.

[85] INFORM Inc., (1992). *Fuzzy-166 Processor--The Chip for Flexible High-Performance Fuzzy Solutions.* INFORM GmbH, Aachen, Germany.

[86] Wang, J. (1993). A 12-bit fuzzy computational acceleration (FCA) core. *Proc. Fuzzy Logic '93*, M332.

[87] Sanchez, E. (1986). Medical applications with fuzzy sets. A. Jones, A. Kaufmann and H.-J. Zimmermann (eds.), *Fuzzy Set Theory and Applications*, D. Reidel, Boston, MA.

[88] Shimizu, K., Osumi, M., & Imae, F. (1992). The digital fuzzy processor FP-5000. *Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks,* 539-542.

[89] Ginart, A., & Sanchez, G. (2002). Fast Defuzzification Method Based on Centroid Estimation. *Applied Modelling and Simulation.*

[90] Saetieo, S., & Torrey, D. (1998). Fuzzy logic control of a space-vector PWM current regulator for three-phase power converters. *IEEE Transactions on Power Electronics*, *13*(*3*), 419 - 426.

[91] Eisele, M., Hentschel, K., & Kunemund, T. (1994). Hardware realization of fast defuzzification by adaptive integration. *Proceedings of the Fourth*

International Conference on Microelectronics for Neural Networks and Fuzzy Systems, 318-323.

[92] Runkler, T., & Glesner, M. (1994). DECADE—fast centroid approximation defuzzification for real time fuzzy control applications. *Proceedings of the 1994 ACM symposium on Applied computing*, 161 - 165.

[93] Saletic, D., Velasevic, D., & Mastorakis, N. (2002). Analysis of Basic Defuzzification Techniques. *Proceedings of the 6th WSES International Multiconference on Circuits, Systems, Communications and Computers*.

[94] Tamukoh, H., Horio, K., & Yamakawa, T. (2007). A bit-shifting-based fuzzy inference for self-organizing relationship (SOR) network. *IEICE Electronics Express*, *4*(*2*), 60-65.

[95] Tamukoh, H., Horio, K., & Yamakawa, T. (2006). A Digital Hardware Architecture of Self-Organizing Relationship (SOR) Network. *Lecture Notes in Computer Science*, Springer, *4234*, 1168-1177.

[96] Lancaster, S., & Wierman, M. (2003). Empirical study of defuzzification. *22nd International Conference of the North American Fuzzy Information Processing Society*, 121- 126.

[97] Patel, A. (2004). Transformation functions for trapezoidal membership functions. *International Journal of Computational Cognition*, *2*(*3*), 115 - 135.

[98] Deliparaschos, K., Nenedakis, F., & Tzafestas, S. (2005). A fast digital fuzzy logic controller: FPGA design and implementation. *10th IEEE Conference on Emerging Technologies and Factory Automation*, *1*, 259 - 262.

[99] Broekhoven, E., & Baets, B. (2006). Fast and accurate center of gravity defuzzification of fuzzy system outputs defined on trapezoidal fuzzy partitions. *Fuzzy Sets and Systems*, *157*(*7*), 904-918.

[100] Zimmermann, H. (1999). *Practical Applications of Fuzzy Technologies (The Handbooks of Fuzzy Sets)*. New York: Springer.

[101] Lee, S., Miyazaki, M., & Kim, J. (2009). Design of Very High-Speed Integer Fuzzy Controller Without Multiplications by Using VHDL. *Lecture Notes in Computer Science, Springer, 4692,* 93-100.