HIGH PERFORMANCE RECONFIGURABLE FUZZY LOGIC DEVICE FOR MEDICAL RISK EVALUATION

by

Kingsley Adeoye

BS in Computer Engineering,

Stony Brook University, 2007

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science in Electrical Engineering

University of Pittsburgh

2009

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Kingsley Adeoye

It was defended on

September 18, 2009

and approved by

Thesis Advisor: Allen C. Cheng, PhD, Assistant Professor,

Electrical and Computer Engineering Department

Steven P. Levitan, PhD, Professor, Electrical and Computer

Engineering Department

Zhi-Hong Mao, PhD, Assistant Professor, Electrical and Computer

Engineering Department

HIGH PERFORMANCE RECONFIGURABLE FUZZY LOGIC DEVICE FOR MEDICAL RISK EVALUATION

Kingsley Adeoye, M.S.
University of Pittsburgh, 2009

To date cardiovascular diseases (CVD) account for approximately 35% of all deaths worldwide. Many of these deaths are preventable if the risk of developing them can be accurately assessed early. Medical devices in use today cannot determine a patient's risk of developing a CVD condition. If accurate risk assessment was readily available to doctors, they can track rising trends in risk levels and recommend preventative measures for their patients. If patients had this risk assessment information before symptoms developed or life-threatening conditions occurred, they can contact their doctors to inquire about recommendations or seek help in emergency situations.

This thesis research proposes the idea of using evolutionary programmed and tuned fuzzy logic controllers to diagnose a patient's risk of developing a CVD condition. The specific aim of this research seeks to advance the flexibility and functionality of fuzzy logic systems without sacrificing high speed and low resource utilization. The proposed system can be broken down into two layers. The bottom layer contains the

controller that implements the fuzzy logic model and calculates the patient's risk of developing a CVD. The controller is designed in a context switchable hardware architecture the can be reconfigured to assess the risk of different CVD diseases. The top layer implements the evolutionary genetic algorithm in software, which configures the fuzzy parameters that optimize the behavior of the controller. The current implementation inputs patient's personal data such as electrocardiogram (ECG) wave features, age and body mass index (BMI) and outputs a risk percentage for Sinus Bradycardia (SB), a common cardiac arrhythmia.

We validated this system via Matlab and Modelsim simulations and built the first prototype on a Xilinx Virtex-5 FPGA platform. Experimental results show that this 3-input-1-output fuzzy controller with 5 fuzzy sets per variable and 125 rule propositions produces results within an interval of approximately 1μs while reducing hardware resource utilization by at least 25% when compared with existing designs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

1.0 - INTRODUCTION

For hundreds of years, the role of healthcare has been a very important part of human society. In the past half century, computer technology has also had a dramatic impact on every part of our society as well. Despite their tremendous contributions, healthcare and computer technology have only recently started to integrate with each other. Various devices like MRIs, heart monitors, X-ray machines and defibrillators have given doctors the ability to detect, diagnose and prevent various diseases in many patients. These devices provide doctors with information that helps them make accurate determinations about a patient's risk of a particular disease.

In the near future, it is highly probable that the next generation of medical devices will be able to assist doctors in making diagnostic decisions in and out of a hospital. Mobile devices such as smart phones can run diagnostic applications to provide information to local hospitals via wireless networks. These mobile devices will not only track and record medical data, but also formulate a preliminary diagnosis for the doctor and patient. In order to develop a device capable of making these types of decisions, it is very important that it be able to "think" and "reason" like a human doctor. Fuzzy logic [1] is one of the best techniques to satisfy this requirement.

Fuzzy logic can implement a computerized model of human reasoning needed to present a medical diagnosis.

By utilizing fuzzy logic's tolerance of imperfect degrees of truth [1], electronic devices can combine computer processing power with human reasoning. In recent years, engineers have been able to develop fuzzy logic systems that classify whether a person has a disease based on specific patient data. The purpose of this research is to develop a device that can not only classify but also calculate a patient's risk of having a particular disease. This type of technology can assist doctors in lowering a patient's risk of contracting these diseases before they become a serious problem which can ultimately save lives. The study focuses on detecting various cardiovascular diseases (CVD) based on electrocardiogram (ECG) wave features in combination with other patient specific information such as patient age and body-mass index (BMI).

1.1 - MOTIVATION

There are many reasons for exploring new ways to innovate medical technology. Technology has enhanced the quality of life for so many people in the modern world. Doctors and nurses have been able to use technology to improve, save and recover many lives. X-ray machines are able to provide information about our skeletal system, while computed tomography (CT) scanners display pictures of our brain, lungs, heart and more. ECG monitoring devices show the rate at which our heart beats, while regular

computer servers allow us to store and protect millions of electronic health records.

Even with all of these advancements in medical technology, much more can be done to provide additional assistance to healthcare professionals. Technology can help hospitals and clinics run more efficiently in a number of ways. To avoid long lines at hospitals and long waits for test results, portable devices can collect a patient's information in real-time before they arrive at the hospital. The devices can also provide preliminary diagnostic results from a remote location and possibly transmit data to hospitals in case a critical emergency care is needed. Such a device can increase a patient's chance of receiving care in an emergency situation, reduce the wait time to diagnose a patient and potentially eliminate expensive tests and procedures.

Over the last decade or so, the cost of healthcare has risen dramatically in the United States [45]. Healthcare is one of the biggest drivers of debt in business, government and everyday homes. A large contributor to this debt is to cover the expense of treating many preventable diseases. Decreasing the number of preventable diseases will provide a huge cost savings to the system and also save many lives. Many of these preventable diseases are related to the cardiovascular system.

According to the American Heart Association [45], cardiovascular disease claimed over 800,000 lives in 2005 (or 35% of all deaths worldwide). Approximately 17% of these deaths were people under the age of 65 years old. CVD related deaths have decreased about 26% over the past 10 years which has been largely attributed to the rapid advancement of technology. With a heavy focus on prevention, doctors have been able to

use technology to help patients curb their risk of contracting chronic heart diseases.

1.2 - GOALS

Currently, research has been done to craft a way to detect and classify certain cardiac arrhythmias. The goal of this thesis is to develop a device that can assess a patient's risk of having a particular cardiac arrhythmia. A system that can assign risk values for specific heart arrhythmias in patients can help doctors prevent diseases from appearing before it becomes critical. Ultimately, this type of device can affect the way we diagnose and treat patients.

By linking levels of risk to other aspects of healthcare delivery, we can drastically change the way we practice and provide medicine. Risk levels can be used to calculate proper drug dosages, formulate dietary suggestions and devise exercise plans automatically. Also by placing the system on a mobile device, patients can monitor their risk levels and send any important information to their physician. Home care devices can be equipped with this special function to provide aid for elderly and disabled patients. There are so many cost benefits associated with this type of device which can stem from reduced hospital congestion, increased emergency efficiency and decreased mortality rates for CVD.

1.3 - CONTRIBUTIONS

As stated earlier, digital classifiers using fuzzy logic rule-based systems have been used to detect and identify cardiovascular arrhythmias. A fuzzy rule-based system calculates output responses for given combinations of input variables. In our simulation, we attempt to calculate the risk associated with certain arrhythmias. The output response is linked to the percentage risk value while the input variables are linked to quantifiable data found in a patient's medical records. A patient's age, body-mass index and even electro-cardiograph (ECG) signal features like heart-rate, amplitude and period are all examples of quantifiable data.

Our main contribution in this design is the association of risk with the detection of arrhythmias in patients. This paper builds off the fuzzy rule-based system and is optimized using innovative evolutionary techniques. Our fuzzy controller can operate as an accurate medical diagnostic system based on existing clinical evaluations and surveys that are referenced during training. Another contribution we provide is a new fuzzy logic controller hardware architecture based on current models [12, 15, 16, 22 and 34]. We seek to improve system flexibility, timing and area attributes of existing fuzzy logic designs. The third contribution we provide is the optimization of the fuzzy logic controller during the evolutionary training process. We applied a different approach to existing training algorithms allowing for more reliable searches without rapid convergence to local optima. These modifications were important due to significant complexity of developing a practical medical diagnostic risk analyzer.

1.4 - SYSTEM OVERVIEW

In this thesis research, the major task was to design a genetic programming trained fuzzy logic controller implemented in two layers. The bottom layer contains the fuzzy controller hardware, a fully context-switchable system prototyped on a FPGA hardware device. It is designed to accept up to three 8-bit input variables and display one 8-bit output. The switchable contexts in the fuzzy controller block are the fuzzy membership graphs for each input variable and the rule matrix, which will be defined in later sections. This provides flexibility and added capability of operating on more than one disease or input medical data. The top layer is an iterative evolutionary algorithm implemented in MATLAB software that optimizes the switchable parameters for the fuzzy controller. Existing clinical data found in various medical surveys and evaluations provide the algorithm with guidance to train our controller. The goal of this training is to search for the best fuzzy logic parameters possible that enables the controller to mimic the judgment and reasoning behind the evaluations performed in existing medical surveys.

1.5 - THESIS ORGANIZATION

This section provides an overview and brief description of the material covered in this thesis. Chapter 2 introduces fuzzy set theory, fuzzy logic control and various models that have been conceptualized over the

years. This chapter also discusses the purpose of using evolutionary genetic algorithms and various methods for implementing the algorithm. Chapter 3 reviews existing hardware techniques used to implement various models onto a FPGA device. It also presents multiple hardware and software genetic programs that are used for different applications ranging from robotics to medical diagnostics. We also introduce an innovative genetically based optimization technique for optimizing fuzzy logic systems. Chapter 4 lays out the hardware and software systems being developed in this thesis. The various features of our hardware design as well as the effectiveness of our software training methods are compared with other existing models. Chapter 5 explains the setup of the simulations and experiments that were conducted and displays the results of these tests along with diagrams and demos. Chapter 6 will conclude this thesis with some discussion and remarks for future research.

2.0 - BACKGROUND

This chapter will introduce two important concepts related to the research project undertaken in this paper. Fuzzy logic control and evolutionary algorithms are very intelligent methods of combining "real world" abstraction with "virtual world" computer power. The goal of both techniques is to apply concepts found in natural phenomenon and basic reasoning to effective and high speed digital technology. This approach will permit technology to "think" and "learn" in the abstract and assist humanity with many additional objectives and functions other than conventional data storage and retrieval. We will begin with fuzzy logic and continue onto evolutionary algorithms.

2.1 - FUZZY LOGIC CONTROL

2.1.1 - Fuzzy Set Theory

In 1965, Dr. Lotfi A. Zadeh pioneered a revolutionary idea that sought to expand on classical set theory, which dealt with distinct and precise

boundaries of inclusion, to a new fuzzy set theory that dealt with uncertainty and imprecision [1]. In classical set theory, a set contains members that completely belong to that particular and no other set. Fuzzy set theory states that a set can have members that partially belong to that set while also partially belonging to another set. Their level of membership to a fuzzy set is determined by a degree of membership (or degree of truth). This philosophy of ambiguous membership allows for imprecise truths. The main contribution of fuzzy theory is the ability to classify truth with a level of uncertainty, which is perfectly consistent with the logic of human reasoning.

In the real world, logic is often imperfect. For example, a set that is defined as a class of "tall" men does not fit the classical form of a set since not all tall men are exactly the same height. Although this amount of imprecision exists in the real world, humans still make judgments based on this type of information every day. The framework of fuzzy set theory allows for the implementation of problem solving algorithms designed to handle loosely defined criteria for class membership.

A fuzzy set, A, is a set which associates a grade of membership, *f*, with a generalized element, *x*, in a universe of discourse, X. The universe of discourse represents a range of real, or crisp, numbers used to measure a quantifiable attribute. Quantifiable attributes can include things like weight, height, speed, distance or temperature. The grade of membership is a real number between the values of 0, representing no membership, and 1, representing complete membership. For height, some measurements will belong to a fuzzy set "tall" more than other measurements, so they will have a higher grade of membership. An example of fuzzy set A is shown below.

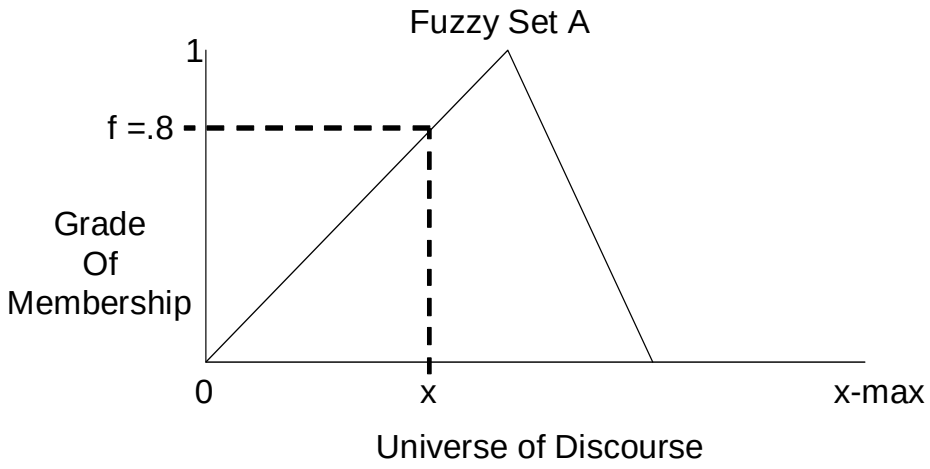

Figure 2.1– Fuzzy Set A with Input x

$$f_A(x) = [0, 1], \qquad x \in X$$

A fuzzy membership function is a graphical representation of a fuzzy set. These functions can be formulated using linear, exponential or Gaussian equations, plotting grades of membership across a universe of discourse. For example, height is a feature that can be measured in inches or meters, however, humans use words like “tall”, “average”, and “short” to classify height. These words are perfect examples of how crisp and precise values are represented in fuzzy and imprecise linguistic terms. The universe of discourse for height can range from 0 inches to 96 inches, or 8 feet. Each grade of membership scores how much each value *x* belongs to a given fuzzy set ranging from 0 to 1. The fuzzy membership functions and their graphical plots for variable *height* are shown below.

Short:

$$f_{Short}(x) = 1, \qquad 0 < x <= 48$$

$$f_{Short}(x) = -(1 / 18) * x + 3.66, \qquad 48 < x <= 66$$

$$f_{Short}(x) = 0, \qquad 66 < x <= 96$$

Equation 2.1 – Fuzzy Membership Functions for Short Variable

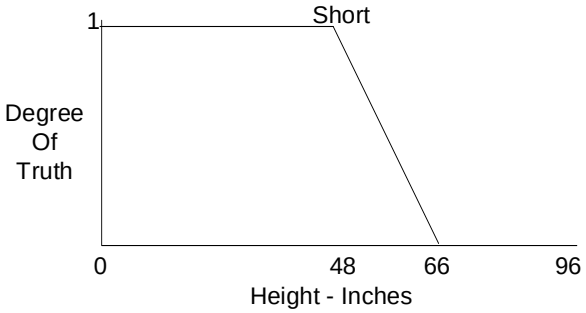

Figure 2.2 – Fuzzy Membership Function Plot for Short

Average:

$$f_{Average}(x) = 0, \qquad 0 < x <= 48$$

$$f_{Average}(x) = (1 / 18) * x - 2.66, \qquad 48 < x <= 66$$

$$f_{Average}(x) = -(1 / 6)*x + 12, \qquad 66 < x <= 72$$

$$f_{Average}(x) = 0, \qquad 72 < x <= 96$$

Equation 2.2 – Fuzzy Membership Functions for Average Variable

Figure 2.3 – Fuzzy Membership Function Plot for Average

$$\textbf{Tall:} \quad f_{Tall}(x) = 0, \qquad 0 < x <= 66$$
$$f_{Tall}(x) = (1/6)*x - 11, \qquad 66 < x <= 72$$
$$f_{Tall}(x) = 1, \qquad 72 < x <= 96$$

Equation 2.3 – Fuzzy Membership Functions for Tall Variable

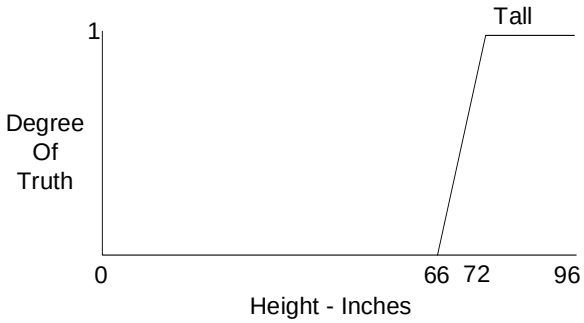

Figure 2.4 – Fuzzy Membership Function Plot for Tall

A fuzzy membership graph plots the fuzzy membership functions for each fuzzy set across the universe of discourse. For example, the fuzzy

representation for *height* can be graphed by plotting the membership functions on a single plane. The x-axis is the universe of discourse, while the y-axis represents the degree of membership.

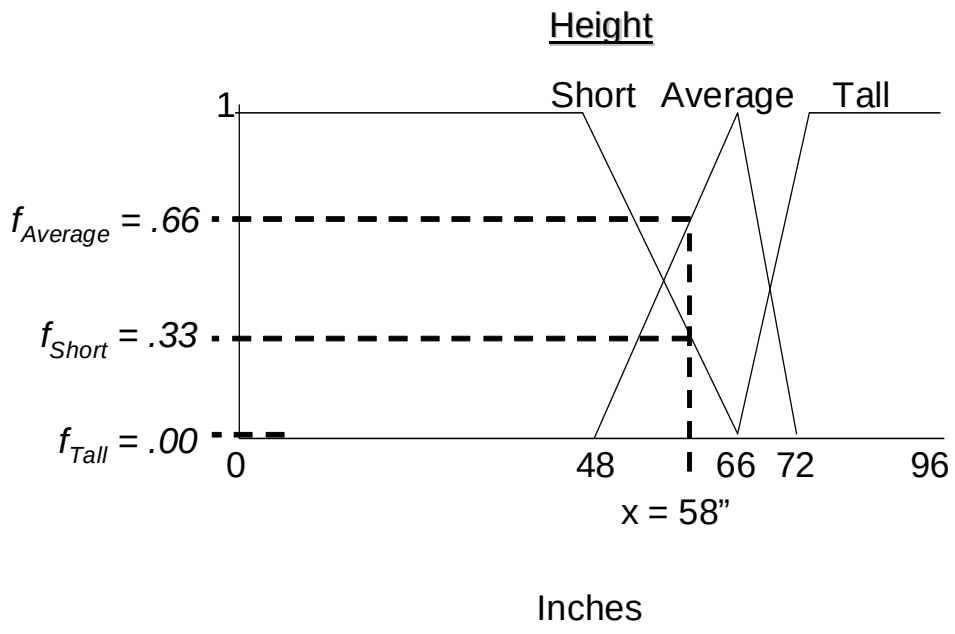

Figure 2.5 – Fuzzy Membership Graph for Height w/ input = 58”

This simple mechanism allows for the classification of crisp and precise elements into imprecise fuzzy terms with variable degrees of accuracy and truth. Each crisp element can produce multiple fuzzy terms with variable degrees of truth depending on the number of overlapping functions. This process is called fuzzification.

Fuzzy set theory also incorporates many of the operations and properties found in classical set theory. Union, intersection, complementation and algebraic arithmetic are some examples of operations that can be performed on fuzzy sets. Fundamental properties such as

association, distribution and De Morgan's law are also held by fuzzy sets. An example of each of these operations and properties are shown below.

Fuzzy Set Operations

Union: $f_{or}(x) = f_A \vee f_B = \text{Max}\,[f_A(x), f_B(x)], \quad x \in X$

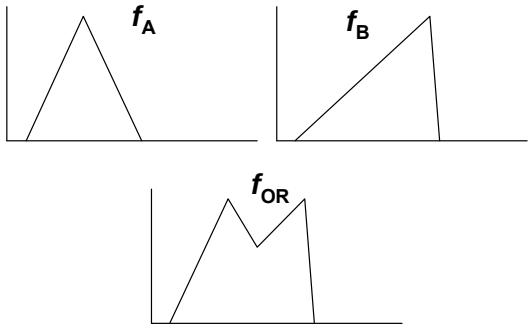

Figure 2.6 – Union Fuzzy Operation

Intersection: $f_{and}(x) = f_A \wedge f_B = \text{Min}\,[f_A(x), f_B(x)], \quad x \in X$

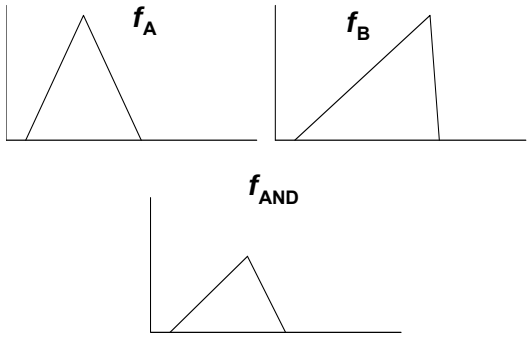

Figure 2.7 – Intersection Fuzzy Operation

Complement: $f_{not}(x) = 1 - f_A$
$= 1 - f_A(x), \quad x \in X$

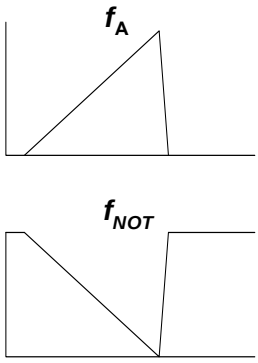

Figure 2.8 – Complement Fuzzy Operation

Fuzzy Set Properties

Association: $(f_A \vee f_B) \vee f_C = f_A (f_B \vee f_C)$
$(f_A \wedge f_B) \wedge f_C = f_A (f_B \wedge f_C)$

Distribution: $(f_A \vee f_B) \wedge f_C = (f_A \wedge f_C) \vee (f_B \wedge f_C)$
$(f_A \wedge f_B) \vee f_C = (f_A \vee f_C) \wedge (f_B \vee f_C)$

De Morgans: $(f_A \vee f_B)' = f_A' \wedge f_B'$
$(f_A \wedge f_B)' = f_A' \vee f_B'$

These operations and properties are very important to the design and development of fuzzy logic control, which is discussed in full detail in the next section. Also, several examples from different disciplines will be presented to outline the fundamental importance of fuzzy logic in modern technology.

2.1.2 - Fuzzy Logic

Traditionally, mathematical formulas have provided very accurate and predictable calculations for control design. However, engineers have a difficult time applying them to real world problems with non-linear dimensions [7]. For this reason, fuzzy logic control has become an attractive scheme for developing complex controllers. Dr. Ebrahim H. Mamdani first introduced the concept of fuzzy logic control by expanding on fuzzy set theory [3]. Fuzzy logic control combines the concept of imprecise knowledge from fuzzy set theory with computational logic from classical set theory. Therefore, a controller can make decisions based on imperfect information through classical logic operations.

By using linguistic terms and readable words, fuzzy logic simplifies the deductive reasoning process in many control applications [2]. It allows engineers to design decision-making systems that operate based on if-then rules. Each rule proposition contains two parts: an antecedent and a consequent. The antecedent is a condition for the rule proposition, while the consequent is the conclusion of that proposition. These types of rules can also be assigned to many human-centric fields such as economics, linguistics, law, psychology and medicine [6].

Rule Proposition

IF Antecedent is TRUE **-** **THEN** Consequent is TRUE

Mamdani's work on fuzzy controllers has shown that based on empirical knowledge of the physical system, rules can be implemented effectively as a basis for operation. Based on his contributions, many current systems are controlled using fuzzy logic. Some examples of these

systems include home appliances, automotive technology, industrial machinery and many other electronic devices.

An automatic braking system is a perfect example of how control systems can be implemented using fuzzy logic. The various factors that must be considered in the design include the car's speed, acceleration and distance of the closest object ahead of the car. The objective of this control system is to determine the right amount of pressure to apply to the brakes. Normally, engineers would attempt to formulate mathematical equations that determine the proper brake pressure as a function of the speed, acceleration and distance. However, fuzzy logic allows the engineer to develop a set of linguistic rule propositions based on expert knowledge of the physical braking system.

Fuzzy logic control can be broken down into three main phases. The following figure displays the execution flow for a fuzzy-based automatic braking system.

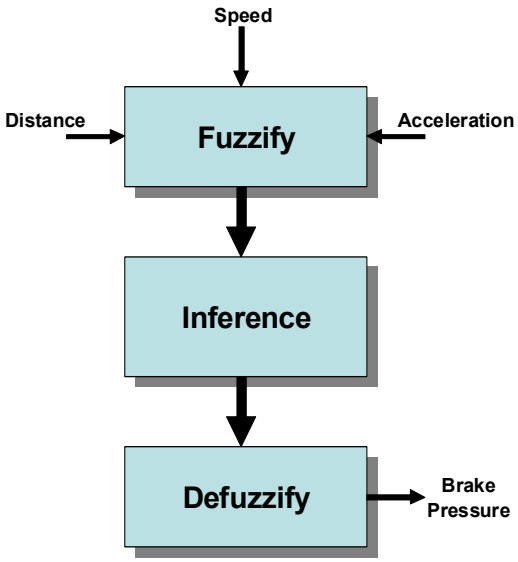

Figure 2.9 – Fuzzy Operation Execution Flow

The first phase, fuzzification, reads the crisp input values and identifies the appropriate fuzzy terms and degrees of membership. The fuzzy membership functions and graphs are very important to the classification of quantifiable variables like speed, acceleration and distance.

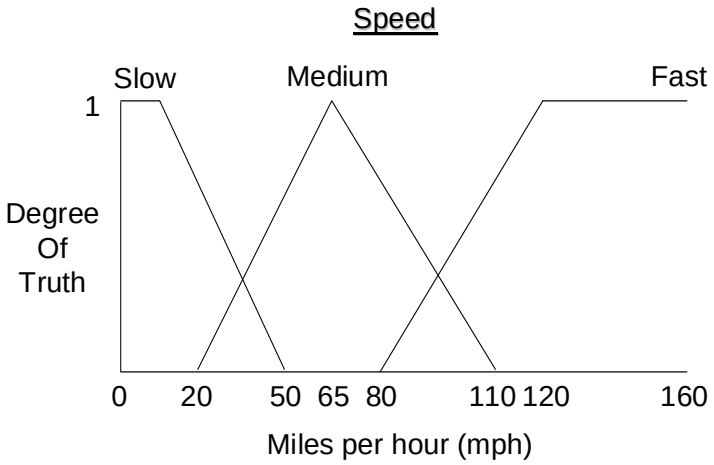

Figure 2.10 – Fuzzy Membership Graph for Speed

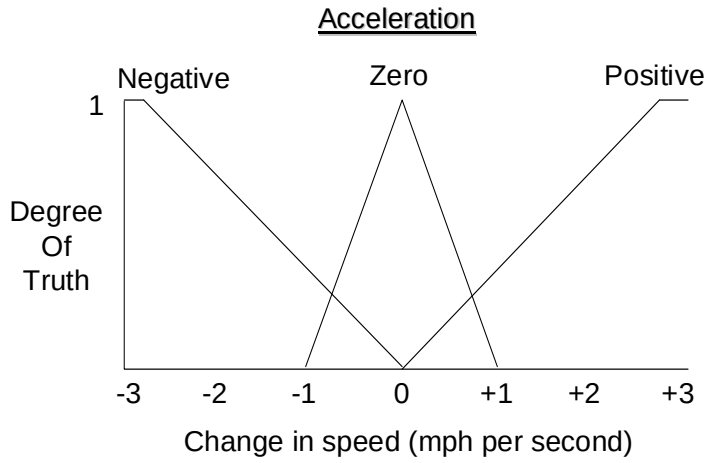

Figure 2.11 – Fuzzy Membership Graph for Acceleration

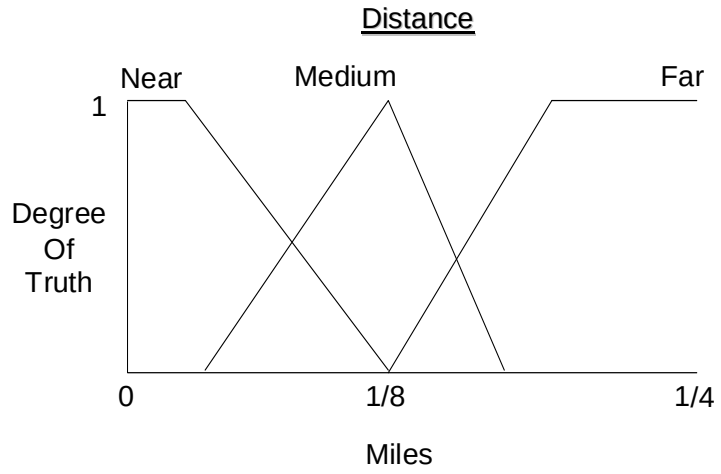

Figure 2.12 – Fuzzy Membership Graph for Distance

This process determines the parameters needed for the next phase. The figure below shows the input and outputs of a fuzzification block, where each input requires an individual fuzzifier. The number of fuzzy term-degree pairs is equal to the number of fuzzy sets included in the design by the engineer.

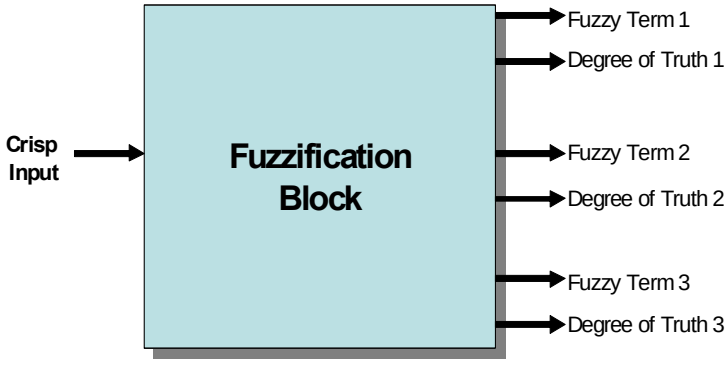

Figure 2.13 – Fuzzification Block

The inference phase is the most crucial part of fuzzy logic. This stage contains the rule propositions designed to incorporate all of the expert

knowledge about the physical system being controlled. The inference block is essentially a table of if-then rules that link linguistic terms for input variables to linguistic terms for output variables. Output fuzzy terms and their degrees of truth are selected and derived based on specific combinations of input fuzzy terms and their degrees of truth. When the antecedent of a rule is true, then it is labeled active and "fired", otherwise, the rule is labeled inactive and "not fired". The consequent degree of truth for that rule is called the firing strength. An engineer can design the rule matrix accordingly to effectively manage the operation of the controller. The size of the rule matrix depends on the number of system inputs and different fuzzy sets representing those inputs.

$$\text{No. of rules} = \Pi\,(\text{No. of fuzzy sets for Input } k), \qquad 1 < k < \text{No. of inputs}$$

Equation 2.4 - Formula: No. of Rules

As mentioned before, the automatic braking system requires inputs for speed, acceleration and distance from object, each having a fuzzy membership graph with three fuzzy sets. The figure below displays a rule matrix for an automatic braking system.

Input Categories			***Output Categories***
Speed	Acceleration	Distance	Brake Pressure
Slow	Negative	Near	Small
Medium	Zero	Medium	Moderate
Fast	Positive	Far	Large

Rule Example

IF Speed = Slow **and** **IF** Acceleration = Negative **and** **IF** Distance = Near

THEN Pressure = Large

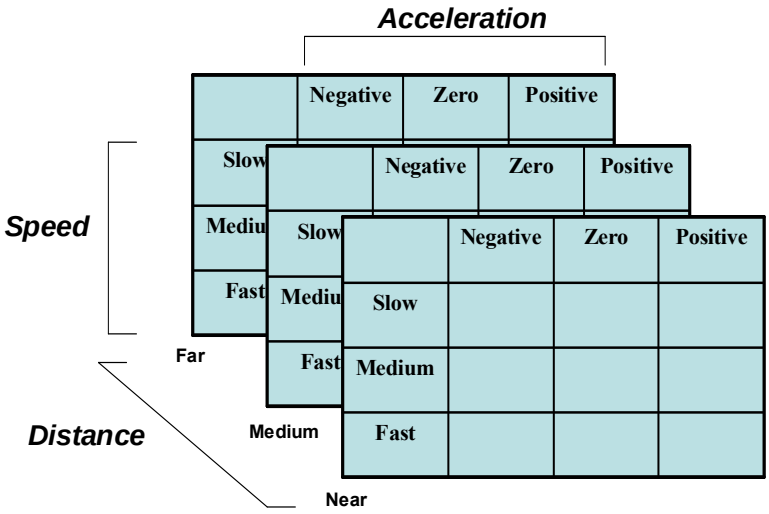

Figure 2.14 – Autobraking System Rule Matrix

The matrix allows for every possible combination of inputs fuzzy terms to be considered. The elements of the matrix will be filled with the fuzzy terms for brake pressure, which include "small", "moderate" and "large". The matrix must be devised using expert knowledge to ensure that the controller is functional accurate. The fuzzy membership graph for pressure is shown.

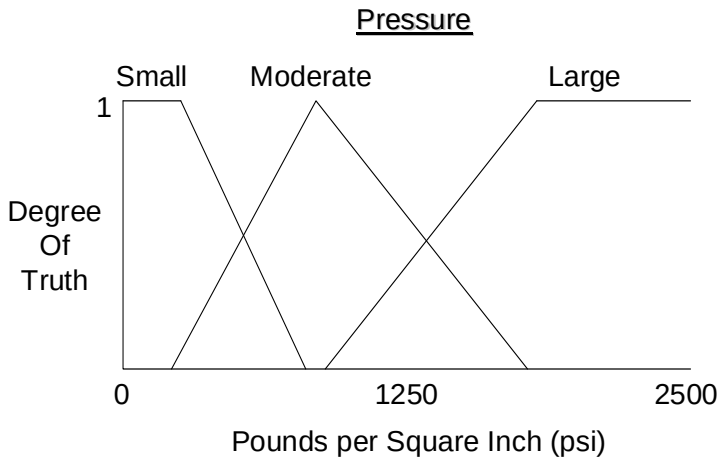

Figure 2.15 – Fuzzy Membership Graph for Pressure

Different methods exist for calculating the output fuzzy term degrees. The more common method is described below.

Step 1. Determine each rule's firing strength

$$fs\ (\text{Rule 1}) = \text{MIN}(f_{\text{Slow}}(x), f_{\text{Negative}}(y), f_{\text{Near}}(z))$$

$$fs\ (\text{Rule 2}) = \text{MIN}(f_{\text{Medium}}(x), f_{\text{Negative}}(y), f_{\text{Near}}(z))$$

$$\vdots$$

$$fs\ (\text{Rule 26}) = \text{MIN}(f_{\text{Medium}}(x), f_{\text{Positive}}(y), f_{\text{Far}}(z))$$

$$fs\ (\text{Rule 27}) = \text{MIN}(f_{\text{Fast}}(x), f_{\text{Positive}}(y), f_{\text{Far}}(z))$$

Step 2. Combine the firing strengths according to rule consequents

$$f_{\text{Small}} = \text{MAX}(\text{all } fs\ (\text{Rule N}) \text{ with consequent Small}),\ 1 < N < \#\ \text{of Rules}$$

$$f_{\text{Moderate}} = \text{MAX}(\text{all } fs\ (\text{Rule N}) \text{ with consequent Moderate}),\ 1 < N < \#\ \text{of Rules}$$

$$f_{\text{Large}} = \text{MAX}(\text{all } fs\ (\text{Rule N}) \text{ with consequent Large}),\ 1 < N < \#\ \text{of Rules}$$

The following block diagram displays the inputs and outputs of the inference block based on the automatic braking system model. The number of input fuzzy term-degree pairs and output fuzzy term-degree pairs is dependent on the design. In this model, the inference block inputs nine fuzzy term-degree pairs and outputs three fuzzy term-degree pairs.

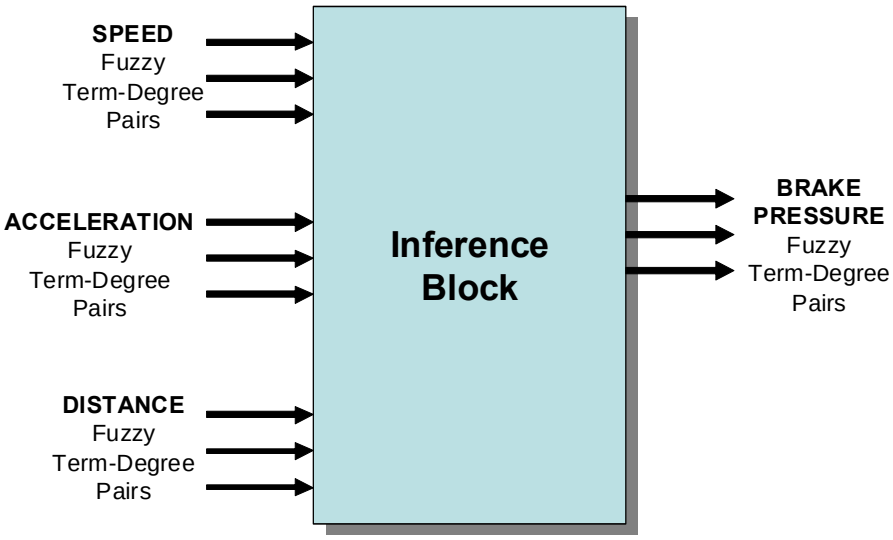

Figure 2.16 – Inference Block

The resulting fuzzy term and degree pairs for the output variable are very important for the next and final phase, defuzzification. The defuzzification phase is responsible for calculating a crisp value from the output linguistic fuzzy terms and degrees. This phase is essentially the reverse of the fuzzification process, allowing the fuzzy linguistic terms to be converted to a crisp number. For the automatic braking system, the crisp number represents the unit of pressure applied to the brakes. The block diagram for defuzzification is shown below.

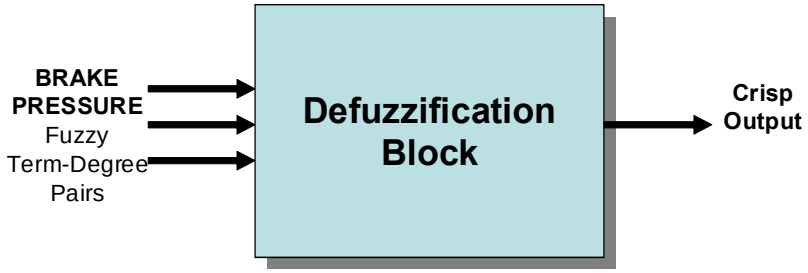

Figure 2.17 – Defuzzification Block

Defuzzification has two common approaches in the conventional fuzzy model: center of area and center of gravity functions. The center of area function is more accurate but requires increased computations; while the center of gravity function is less accurate but with far less computations.

The center of Area function calculates the crisp output number, Z_C, using the Mass and Area of the aggregated shape of the output fuzzy membership graph.

Center of Area

$$Z_C = \frac{\textbf{MASS}}{\textbf{AREA}} = \frac{\int j * f\text{-}aggregate\ (j)}{\int f\text{-}aggregate\ (j)}$$

Equation 2.5 – Center of Area Formula

An example of the *pressure* output fuzzy membership graph with their degrees of truth sent from the inference phase is shown below.

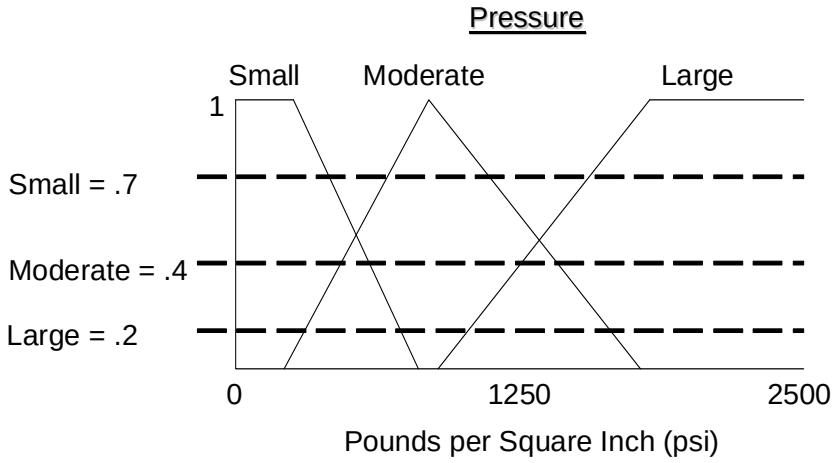

Figure 2.18 – Output Degrees of Truth for each Fuzzy Set

Each output fuzzy set is manipulated based on the degree of truth received from the inference phase.

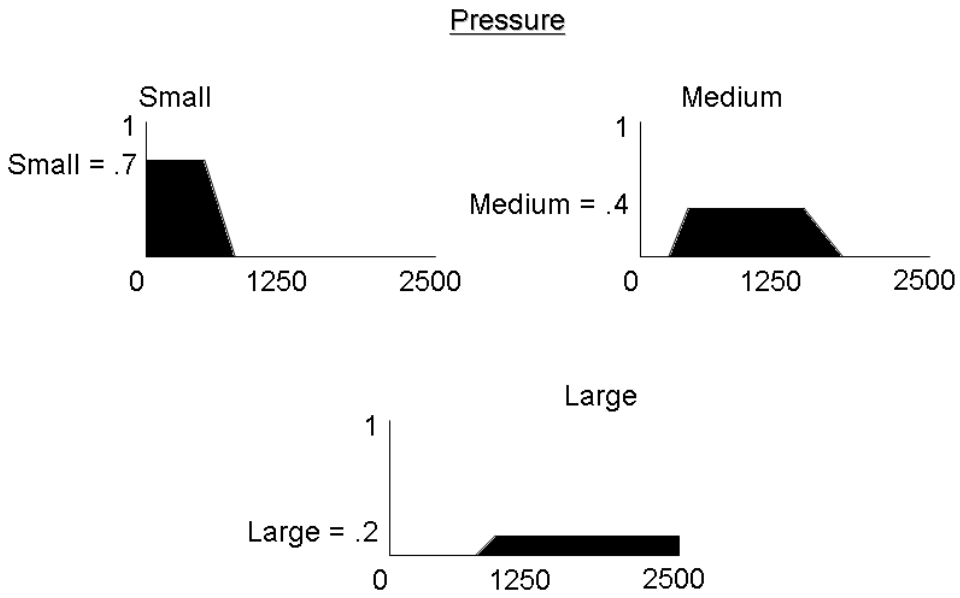

Figure 2.19 – Manipulated Fuzzy Sets in Aggregate Process

The aggregated fuzzy graph is formed by using the MAX function on these manipulated fuzzy sets across the output universe of discourse, U.

Fuzzy Aggregated Sets

$$f\text{-}agg\ (\ u\ ) = \text{MAX}\ (\ f_{\text{Small}}{}^{*}\ (\ u\ ),\ f_{\text{Moderate}}{}^{*}\ (\ u\ ),\ f_{\text{Large}}{}^{*}\ (\ u\ )\ )\ ,\ u \in U$$

Equation 2.6 – Fuzzy Aggregation Logic

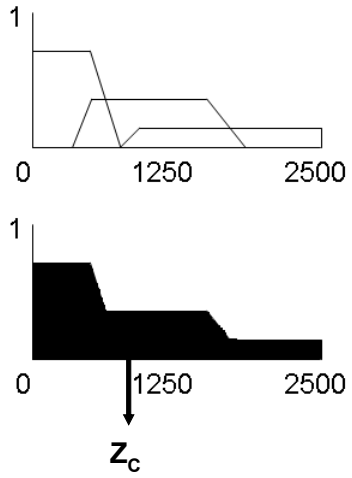

Figure 2.20 – Center of Area Defuzzification w/ Output Zc

The center of gravity function provides a less accurate estimation of the calculated crisp output. The equation focuses on using the center of gravity (COG) crisp output value for each individual fuzzy set and their corresponding degrees of truth.

Center of Gravity

$$Z\text{-}c = \frac{\sum U_{COG} * f(U_{COG})}{\sum f(U_{COG})}$$

Equation 2.7 – Center of Gravity Formula

The following diagram displays how the formula is calculated.

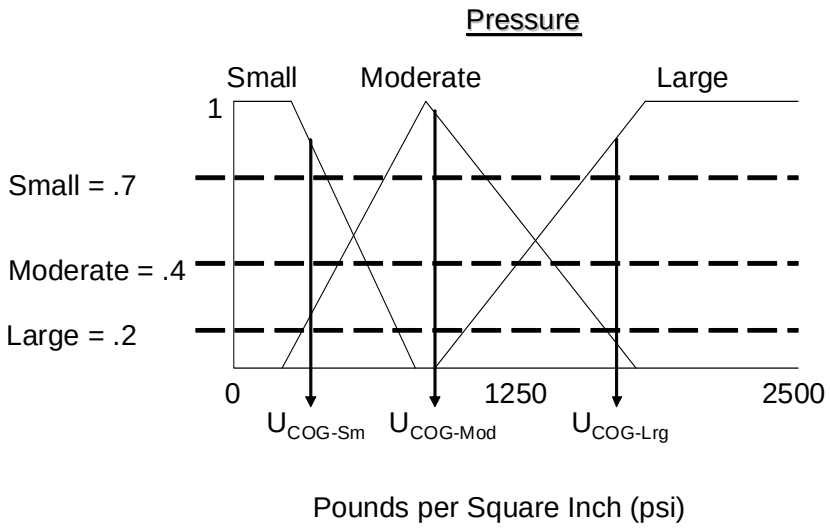

Figure 2.21 – Centers of Gravity for each Fuzzy Set

$$Z_C = \frac{U_{COG\text{-}Sm} * f(U_{COG\text{-}Sm}) + U_{COG\text{-}Mod} * f(U_{COG\text{-}Mod}) + U_{COG\text{-}Lrg} * f(U_{COG\text{-}Lrg})}{f(U_{COG\text{-}Sm}) + f(U_{COG\text{-}Mod}) + f(U_{COG\text{-}Lrg})}$$

Equation 2.8 – Expanded Center of Gravity Formula

The next section will present several types of modeling techniques of fuzzy logic systems. Each technique offers different advantages varying from stability to adaptability to cost effectiveness. Focus and preference will be given to the technique that best fulfills the design goal of this research paper.

2.1.3 - Fuzzy Modeling

Fuzzy-based computer systems have been thoroughly investigated over the past four decades. As a result, many different designs and architectures have emerged providing many additional advantages. These models are often developed based on the application that the system is intended for. Some applications require memory-intensive, non-real time, linguistic-based systems while others require computation-intensive, real time, numeric-based systems. Fuzzy logic has had a tremendous amount of success in many different areas especially those with non-linear dimensions and non-analytic features.

In 2006, Dr. Gang Feng completed a comprehensive survey on fuzzy control systems which outlines the gains and benefits of different categories of fuzzy-based models [8]. There are several generations of fuzzy models that will be covered in this section categorized by the implementation of membership functions, rule generation and defuzzification methods. The various categories that are the most firmly associated with our research topic include:

- Conventional Fuzzy Control
- Neuro Fuzzy Control
- Adaptive Fuzzy Control
- Takagi-Sugeno-Kang Fuzzy Control

Conventional fuzzy control systems refer to the first generation modeling technique. This model follows the Mamdani approach to implementing the fuzzy logic algorithm. Fuzzification and inference phases are setup using expert knowledge of the input variables and operation of the

physical system. The fuzzy membership graphs and rule matrix are used to design these stages. The defuzzification process can follow either center of gravity or center of area functions based on the fuzzy membership graph of the output variables. As stated earlier, these methods are the most popular techniques and have been designed for many control systems [40].

Although conventional fuzzy models are based on the fundamental steps of fuzzy logic, there are many setbacks in this methodology. The first disadvantage is the demand for expert knowledge required to formulate the shape of the membership graphs and elements of the rule matrix. The paradigm assumes that these features are provided by engineers with proficient comprehension of the physical system. This information is not always readily available. Further analysis of conventional models has also shown that stability is a major problem in control design applications [8]. In real-time and safety-critical control problems, conventional fuzzy models do not provide steady responses. Finally, the last piece of difficulty with this particular technique is the computation-intensive nature of the design. The aggregation of membership graphs and the defuzzification of output fuzzy terms require a large number of operations to be performed which increases response delay and design complexity. These issues have led to the creation of new fuzzy modeling techniques over the past several decades.

Neuro-fuzzy and adaptive fuzzy modeling concepts address the issue of high demands for expert knowledge. For many applications, it is very difficult to systematically devise a set of fuzzy membership graphs and rule propositions. For example, different automotive experts may disagree on how to categorize speed, acceleration and distance using fuzzy graph functions. They may also interpret the required rules that govern the controller's operations in different ways. Therefore, developing methods

that can select effective fuzzy logic parameters is very beneficial to fuzzy logic expansion.

Neural networks provide another form of intelligent control with the special ability to acquire knowledge through dataset training [9]. Essentially, neural networks can "learn" how to operate without expert knowledge of the physical system, as long as the provided dataset fully represents the domain of interest. However, the disadvantage of neural network is its heavy dependence on mathematical formulas which is not easily interpreted by humans. The combination of the fuzzy logic and neural networks can provide the advantages of both techniques while eliminating major disadvantages.

Neuro-fuzzy control implements the concept of fuzzy logic over the framework model of neural networks. Neuro-fuzzy modeling solely focuses on the mathematical portion of fuzzy logic algorithm. Engineers are able to use neural nodes to conduct fuzzy operations and basic computations. According to the ANFIS model [9], fuzzy neural nets can be broken down into five layers: input, fuzzify/antecedent, rules/normalization, consequent and aggregate/output layer.

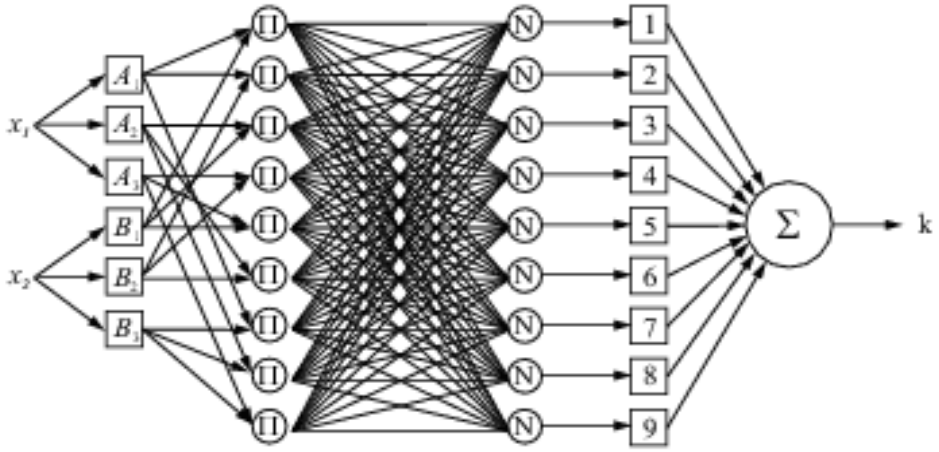

Figure 2.22 – Artificial Neural Fuzzy Inference System

The fuzzify-antecedent layer implements fuzzy membership graphs by including a node for each fuzzy set. The rules-normalization layer implements the rule matrix by creating a node for each rule proposition. The consequent and aggregate-output layers implement the center of gravity defuzzification function. The connection weights in the neural net are directly tied to fuzzy logic parameters that are tuned with back-propagation type learning algorithms. The neural net manipulates these parameters to achieve the best performance based on the accuracy of the output against the training dataset [17].

Adaptive fuzzy control operates in a similar manner as neuro-fuzzy control in the way fuzzy parameters are formulated without expert knowledge. However, adaptive fuzzy modeling attempts to address more complex non-linear systems with unidentified functions. The main goal is to find fuzzy logic parameters that allow the controller to approximate complex unknown functions as much as possible. One method uses state feedback mechanisms based on Lyapunov stability theory to regulate and manipulate

fuzzy parameters for the duration of the physical process [10]. Other approaches use physical plant feedback and a learning module using varying mechanisms to self-adapt and self-organize the fuzzy inference process [41]. Each can provide flexibility and robustness to the controller during real-time operation. Another widely popular approach requires less number of mathematical computations using powerful genetic programming techniques to search for optimal fuzzy parameters that exhibit the best performance possible.

Genetic programming is a type of evolutionary algorithm that searches for an optimal solution using techniques inspired by biological characteristics of human evolution. Genetic algorithms can be used to intelligently search for the best input and output membership and rule parameters that can most accurately approximate a complex system using existing training datasets. The algorithm can create a population of solution candidates carefully encoded to represents these fuzzy parameters. After creating this population, genetic operators and functions simulate the process of natural selection. At the end of this process, the resulting fuzzy logic parameters can be inserted into an optimized fuzzy controller.

Although neuro-fuzzy and adaptive fuzzy modeling techniques have been effective in dealing with the lack of readily available expert knowledge, they have not resolved the issues of stabilizing output responses for real-time applications. The Takagi-Sugeno-Kang (TSK) model attempts to address this issue in their fuzzy model while maintaining adaptable behavior. This method makes revisions to the inference process to achieve this goal; however, in some cases, training may still be needed to help accomplish desired functionality.

TSK fuzzy modeling is designed to be a very dynamic in its application to various control problems [8]. The goal is to approximate complex global nonlinear models by using a collection of local linear models. TSK fuzzy modeling is similar to piecewise linear approximation methods of nonlinear control, which uses linear equations around nominal operating points. In fuzzy logic, these operating points represent the antecedents of the rule propositions that enable smooth connections between the linear functions. These linear functions are plugged into each rule consequent and can be represented using different types of functions, such as state space equations, simple linear functions and constant values. Examples of these rule propositions are displayed below.

Rule Propositions with State Space Equation

Rule n: If x is R AND y is S AND z is T

Then state (t + 1) = A1 * state (t) + B1 * input (t) + *U*

output (t) = C1 * state (t) + D1

Rule Propositions with Multi-linear Equation

Rule n: If x is A AND y is B AND z is C

Then output n = *R* * x + *S* * y + *T* * z + *U*

Rule Propositions with Constants

Rule n: If x is A AND y is B AND z is C

Then output n = *U*

The fuzzification and defuzzification methods are similar to other modeling techniques. Fuzzification is implemented with membership graphs while defuzzification is applied using the center of gravity formula.

This modeling technique provides a methodology for designing nonlinear controllers which can improve stability, performance and design uniformity. Two circumstances that must be taken into consideration are: when the model for system is known and when it is unknown. Designing a controller when a model is available is pretty standard. However, the major difficulty presented with TSK modeling is the identification of specific parameters when the model for control is not known. As a result, researchers have presented methods of training the controller to identify parameter coefficients that can best approximate the designated system such as neural networks and genetic algorithms. Another issue is the inability to present interpretable readable terms that can be read by humans. While the TSK modeling technique can approximate smooth nonlinear systems, it also reduces the decipherable aspect of fuzzy logic fundamentals.

The next section will explore evolutionary algorithms and how they provide very robust mechanisms for developing and optimizing various systems. This will introduce the concept of genetic programming and how they can efficiently search for optimal solutions without sacrificing time and effectiveness.

2.2 – EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are optimization techniques whose functions are based on biological concepts of reproduction and natural selection. These techniques operate as highly efficient search algorithms that select the

best performing solutions using the Darwinian model of "survival of the fittest." Darwin believed that the natural evolution of many living species over numerous generations would constantly produce the finest species. By customizing operators that implement natural processes like reproduction, mutation and selection, evolutionary algorithms can embody the essence of Darwinism. These algorithms provide a very robust and dynamic method of conducting quick and effective search process for design exploration and optimization. There are many different fields that benefit these algorithms such as engineering, economics, political and social sciences, robotics and other fields with artificial intelligence.

2.2.1 - Genetic Programming

A genetic program is an evolutionary-based computer program that performs a set of genetic operations on a population of numerically encoded solutions over the course many generations. The general structure of the algorithm is shown below.

[1] Initialize the population
[2] Evaluate initial population
[3] Begin reproductive cycle
    3.1. Perform competitive selection
    3.2. Apply genetic operators to generate new solutions
    3.3. Evaluate solutions in the population
[4] Repeat step [3] until some convergence criteria is satisfied

The steps are used to follow the natural process of evolution. A population is initialized and evaluated. The selection operator is used to evaluate each individual in the newly initialized population. Based on the evaluation

results, the reproduction phase begins within a given population. According to Darwin, the best individuals with the highest evaluation scores reproduce more often than other individuals with the lowest evaluation scores. The reproductive cycle produces a new population. During this cycle, mutation occurs among certain individuals within this new population.

After the cycle is completed, the population is evaluated again to determine new scores. The reproductive cycle is repeated until a certain condition is met. This condition is based on the commonality of a given set of dominant features. After several cycles of reproduction, dominant features become common among the population. Convergence and maturity can be tested by measuring how many individuals possess these dominant features within a given population. When a threshold is reached, the algorithm terminates and a solution is chosen.

The implementation of this algorithm is predicated on two fundamental requirements. The first requirement is that each individual candidate must be encoded to represent a possible solution. This code is called a chromosome. A chromosome (or genotype) is the abstract representation of each individual (or phenotype). Each human possesses a set of chromosomes which determine their physical features. Therefore, each gene in the chromosome contains information describing the features of the candidate directly affecting their performance. As the length of the chromosome increases, the size of the solution space and algorithm difficulty increases exponentially. The figure below describes a chromosome with eight genes. Each gene can be labeled using numbers from 0 to 9.

8-gene Chromosome

Figure 2.23 – Chromosome with 8 genes which range from 0 to 9

The second fundamental requirement is the need for an evaluation test which provides a score for each solution candidate. For example, an individual can be tested on their physical and mental abilities and consequently provided a score for a specific chromosome. In genetic algorithms, this test is conducted using a fitness function. The fitness function is typically based on a mathematical formula that scores how well a system performs under a selected chromosome solution. These mathematical formulas can measure accuracy, efficiency or effectiveness. The mean square error (MSE) method is commonly used to measure the accuracy of a system when evaluated against a set of N pre-validated data. This is used to test how well a particular chromosome solution returns the predicted and desired results.

Accuracy - Mean Square Error

$$\text{Error} = \frac{1}{N} * \sum_{n=1}^{N} (P_n - M_n)^2$$

Equation 2.9 – Mean Square Error

Error represents the total derived error between predicted values and measured values. This process enables the algorithm to identify and select

top performing chromosome solutions. These two essential requirements are the basis of the natural evolution process simulated by genetic programs.

Genetic operators are also important factors in programming algorithms. As stated earlier, three natural processes are necessary for successful evolution: selection, reproduction and mutation. Each process has a corresponding function operator in genetic programming. To implement the selection process, many different options have been considered which include:

- Roulette Wheel Selection
- Tournament Selection
- Elitist Selection

These techniques express different methods of choosing individuals for reproduction from the population. The roulette wheel method selects candidates based on the normalized proportional fitness of all the candidates in the population. The following table shows an evaluated population of 5 different candidates and their calculated fitness and normalized fitness values. The table also includes a section for expected count which predicts the number of times each candidate will be chosen out of 5 selection attempts.

Table 2.1 – Roulette Wheel Fitness Table

Candidate	*Fitness F*	*Normalize F / Sum F*	*Count F / Avg F*
A	48	.33	1.67
B	10	.07	.35
C	44	.30	1.52
D	24	.17	.83
E	18	.13	.63
Sum	144	1.00	5.00
Average	28.8	.20	1.00
Max	48	.33	1.67

These candidates are organized in a wheel according to their normalized fitness. The figure below uses a table instead of a wheel, but the concept is similar. A random variable between 0 and 100 is generated to choose one of the listed candidates. The higher the fitness value, the more likely it is to be selected.

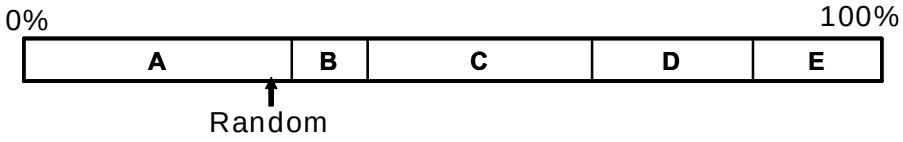

Figure 2.24 – Roulette Wheel w/ Normalized Fitness

The tournament method selects candidates from a subset of individuals from the population through competition. In the example given above, a subset population of 3 individuals can be selected at random to participate in the tournament. Out of the 3 individuals, the best or worst fitted individual can be selected for reproduction depending on genetic

probabilities set by the designer. The elitist method completely bypasses the reproduction phase and transfers a candidate from one generation to the next. Each method is effective in simulating the process of natural selection.

The reproduction operator is another important tool in genetic programming. This phase is responsible for introducing new generations of candidates by combining and mixing two candidates of the previous generation. This is similar to parents having children. The crossover technique is a very common method for simulating this process. Several examples of crossover that exist include:

- One-point Crossover
- Two-point Crossover
- Uniform Crossover

One-point crossover simply states that all the genes beyond a random single point in the parent chromosomes are transferred to their counterparts.

1	3	6	8	4	4	0	3	0	9	5
3	2	4	0	9	7	8	5	6	2	1

1	3	6	8	4	4	8	5	6	2	1
3	2	4	0	9	7	0	3	0	9	5

Figure 2.25 – One-Point Crossover

Two-point crossover transfers genes between two random points in the parent chromosomes to their counterparts.

1	3	6	8	4	4	0	3	0	9	5
3	2	4	0	9	7	8	5	6	2	1

1	3	6	0	9	7	0	3	0	9	5
3	2	4	8	4	4	8	5	6	2	1

Figure 2.26 – Two-Point Crossover

Uniform Crossover uses logical comparisons between two parent chromosomes to determine which genes to transfer. Uniform Crossover is commonly used in binary encoded chromosomes. Two parents are compared with a binary mask to produce two new offspring.

1	0	0	1	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	0	1	1

1	0	0	0	0	0	0	1	0	1	1

0	0	0	1	1	0	1	0	0	0	1
1	1	1	0	1	0	0	1	0	0	0

Figure 2.27 - Uniform Crossover w/ Mask

These techniques offer excellent examples of how reproduction is simulated in genetic programs.

The mutation operator is responsible for introducing randomized genetic alterations that occur in natural evolution. Mutation is a very important factor in nature. Mutation occurs when genes are randomly transformed within a chromosome.

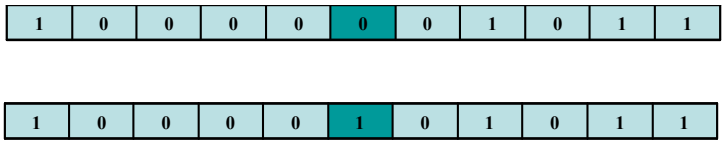

Figure 2.28 – One-Point Mutation Operator

It helps to ensure a level of diversity in the population by inserting small changes between chromosomes and their respective ancestors. Genes can be mutated at one or more points in the chromosome by inverting them directly or through a mask. One-point mutation is illustrated above.

These operators are critical tools in implementing genetic programs. Once all preliminary requirements are met and the code is written, a functional genetic search program can be executed. However, there are several facts that must be considered when observing and studying natural evolution. In nature, reproduction and mutation occurs only a percentage of the time within a given evolutionary cycle. Therefore, reproduction and mutation operators must contain probability settings that demonstrate this concept.

Another important issue that must be resolved is the algorithm's rapid convergence to one solution. This is a major problem with many genetic search programs. In the evolutionary cycle, there are many potential solutions with high fitness ratings. However, the goal is to design a search engine that can find the best solution (or global optimum) over all other possible solutions (or local optima). When a single chromosome becomes too dominant, the algorithm rapidly converges to a local optimum around that chromosome and is mistaken for a global optimum. Rapid convergence eliminates diversity and limits the effectiveness of the algorithm. A large

population size, expanded number of generations, compact-sized chromosome and distinct stopping conditions are useful measures in preventing early convergence and increase search efficiency.

The next chapter will introduce several research papers which focus on the implementation of fuzzy logic controllers. Each design is structured based on one of the modeling techniques outlined in the current section. The various advantages and disadvantages of each design will also be discussed, presenting a baseline study for our proposed fuzzy hardware system. It will also introduce several methods for developing genetic systems using both software and hardware based techniques. The costs and benefits of each approach will be examined and introduced as a baseline for our new training routine.

3.0 – RELATED WORKS

3.1 – FUZZY LOGIC CONTROLLER DESIGN

Many different types of fuzzy modeling techniques have been developed and implemented on engineering platforms. As stated earlier, engineers have relied on fuzzy logic to design controllers for many applications. Fields range from biomedical to heavy industrial systems. This section will outline some examples of fuzzy logic for both software and hardware programs.

Some researchers have demonstrated how fuzzy logic can be applied to software packages as shown in [14]. However, many engineers have attempted to model the fuzzy logic algorithm using hardware design languages. In addition to significant advantages in speed, power consumption and size, FPGA technology allows for flexible implementation and fast prototyping over software and other ASIC models. With this feature, the fuzzy logic operation can be highly parallelized and pipelined. The scale and complexity of the hardware design is usually based on the several parameters which include:

- No. of bits representing the input variables
- No. of bits representing the output variables

- No. of bits representing the membership levels (degree of truth)
- No. of fuzzy terms for inputs and outputs
- Amount of overlap between fuzzy sets within universe of discourse
- Methods for each phase implementation

In [15], Hung and Zajac design fuzzy inference engine using a XILINX 4005PC84-6 FPGA operating at a frequency of 40MHz. The system hosts two 6-bit inputs with 3 fuzzy terms each, one 14-bit output and a 4-bit degree of truth. The fuzzification process is done in parallel using three 64 x 4-bit SRAM modules which store three fuzzy sets for each input. Fuzzy sets are pre-stored in each SRAM. MIN and MAX modules are used to implement the inference phase without a rule matrix. The firing strength from each of the 9 rules is passed through this phase in a parallel network within both modules. A multiplier EPROM and a summation module are used to determine the numerator and denominator of the center of gravity method. Finally a division EPROM retrieves the quotient for the output signal. This approach is very simple to implement and provides fast fuzzy logic operations. Although the design is fully parallelized, there is a heavy reliance on pre-stored multiplier products and division quotients to function limiting the flexibility and dynamics of the design.

In [13], Singh and Rattan developed a similar system with some advantages over [15]. This system hosts two 8-bit inputs and one 8-bit output with 7 fuzzy terms each and an 8-bit degree of truth. As opposed to using memory blocks for fuzzification, they utilize a linear calculator that dynamically assigns the membership degree of truth for each input. Also, the inference process is much more interpretable with the use of a rule memory block containing a rule matrix. However, more controls signals and addition execution time are required since each rule consequents is

processed sequentially. Sequential accumulators use inference data to find a numerator and denominator for the center of gravity method. A division module using an innovative division technique does not rely on pre-stored quotients. This technique provides more rule interpretability and less pre-stored memory. However, the input fuzzifier requires calculations to be performed for each input, which increases computational power consumption.

In [16], Costa, De Gloria and Olivieri focused on developing an asynchronous fuzzy controller based on conventional modeling. Fuzzy logic requires several steps from different phases to be complete in sequence. Therefore, a global synchronous system is needed to design the system in hardware. However, Costa et al explained that the dynamic properties of fuzzy logic can be exploited with the application of local intuition and completion signals. Handshaking signals and custom micro-hardware are used to manage and control the operations between logic blocks and memory registers. Five functionally independent stages are created to provide better performance. The system uses a memory-based fuzzification system, rule matrix inference and center of area method for defuzzification. Three different complexity levels were used to test the functionality of the system based on varying number of inputs, outputs and rules. Simulations confirmed that with higher complexity, asynchronous fuzzy systems provide better performance than synchronous fuzzy systems. The proposed system offers good insight on pipelining and handshaking control management which provide increased coordination between successive fuzzy logic modules. The only disadvantage is that better performance can only be seen in larger, more complex systems.

In [12], Chiueh presents an optimized approach to the design of conventional fuzzy modeling offering an interpretable and fully parallelized structure. The architecture includes memory blocks for fuzzy membership graphs and rule sets along with multiplexer, aggregate and defuzzify blocks separated into pipelined stages. Their design model focuses on two important aspects of fuzzy logic: the overlap between fuzzy sets in a membership graph and the large bottleneck from defuzzification.

In fuzzy logic design, overlaps between fuzzy sets are an important part of the architecture. If the maximum overlap between fuzzy sets is 2, then a fully membership graph can be described using only two memory arrays for any given number of fuzzy sets. The diagram below explains how 3 arrays store membership graph labels for 3 different fuzzy sets.

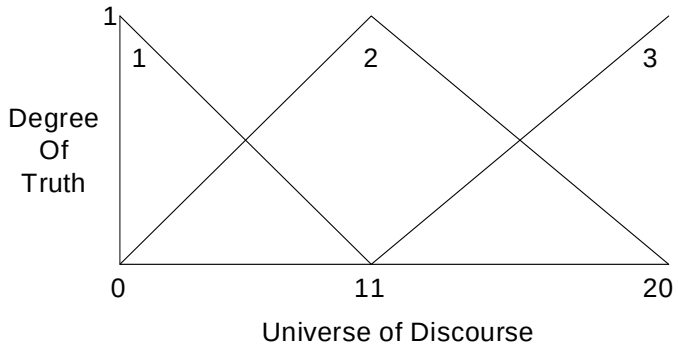

Figure 3.1 - Fuzzy Membership Graph w/ 3 fuzzy sets

1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0	0	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3

Figure 3.2 - Fuzzy Membership Graph Label Memory with 3 Arrays

The following diagram now illustrates how 2 arrays can store the same membership graph labels for the same fuzzy sets. This method can also be used for storing each label's corresponding degrees of truth in memory.

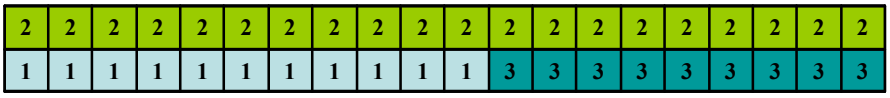

Figure 3.3 - Fuzzy Membership Graph Label Memory with 2 Arrays

Another cost-saving measure, presented in the paper, is also directly tied to fuzzy set overlap. To increase parallelization, a rule memory block is usually created for every possible rule proposition that exists in the rule matrix. This allows every rule to be processed simultaneously. However, since fuzzy set overlap indicates the maximum number of fuzzy sets that can be chosen at any given time, then the maximum number of rules that are activated during the inference process is also fixed.

(a) No. of rules = Π (No. of sets for *F-i*) $1 < i <$ No. of inputs

(b) No. of rules activated = (No. of inputs) ^ overlap

Equation 3.1 – Number of (a) Possible and (b) Active Rules

Fuzzy logic only requires information from the rules that have been activated. As a result, a reduced number of rule memory blocks are needed to parallelize the inference process.

The center of area defuzzification is a very reliable technique in fuzzy logic. On the other hand, it presents a major bottleneck for many fuzzy

controllers forcing many designs to use the less reliable center of gravity method. Calculating the mass and area of a varying aggregate graph requires a great deal of execution time and computational resources. Therefore, [12] focuses on reducing these requirements by using partial sums for mass and area in non-overlapping regions of the aggregated graph that can be pre-calculated and stored in memory for eventual use. This memory stores the mass and area for each fuzzy set graph based on every possible derived output degree of truth. Any region where an overlap exists, the corresponding mass and area are calculated during execution time. Extra memory and control is needed to implement this type of defuzzification but the goal is to eliminate the need for processing certain sections of the aggregated graph during run-time. The main advantage in this model is the ability to determine the area and mass of a graph simultaneously while the main drawback is that the defuzzification method is only beneficial if a large part of the output membership graph is non-overlapping. If only a small portion of the graph was non-overlapping, the partial sum memories would be extremely wasteful.

In [17], Chowdhury et al propose using the neuro-fuzzy modeling technique to develop a fuzzy controller. A three layer expert system is designed to implement the inference and defuzzification process. The weight of each neural connection is determined by training the controller via backpropagation-learning algorithms. The system adapts these weights to the datasets taken from reliable surveys. There are six inputs for the system with three sets for each variable. An external fuzzy interface is responsible for fuzzifying input data using sigmoid functions. The functions are based on variables for center, slope and width of a sigmoid, which are manually tuned to their optimal settings. The neuro-fuzzy model is an extremely

powerful method of training and implementing fuzzy logic; however it does not provide sufficient interpretability of rules and membership for a layman users.

In [11], Ciftcioglu designed a TSK fuzzy controller with a genetic optimization algorithm. The inference phase is implemented using local linear functions. Each rule proposition incorporates a function based on optimized coefficients and system input values. The coefficients are learned through evolutionary training algorithm. The center of gravity method defuzzifies the output variable. This method also does not provide interpretable rule structures that can be reasonably understood since piecewise linear functions are used for rule consequents. This system also requires evolutionary training which provides a means for optimizing the hardware design.

This section offers comprehensive insight into the design and development of fuzzy logic controllers on various hardware and software platforms. Each research paper implements a fuzzy controller based on one of several fuzzy modeling techniques. Architecture flexibility, hardware parallelization and other cost-savings each play important roles in the design of fuzzy controllers. Training algorithms allows engineers to optimize controllers using sets of input-output training data. The next section will discuss how fuzzy controllers apply to medical diagnostic sciences.

3.2 – FUZZY LOGIC IN MEDICAL SCIENCES

There are many different papers that illustrate how fuzzy logic is applied to medical sciences. The decision making capability of fuzzy logic provides a great mechanism for automating the practice of diagnosing diseases. Traditionally, medical decisions have been made by health professionals; however, fuzzy logic systems have shown great potential in approximating these decisions. This section will describe different fuzzy-based approaches that have been developed for automatic diagnostic systems.

Medical diagnostic procedures are performed to identify diseases based on patient physiological data. For this reason, fuzzy classification algorithms are commonly used to identify different diseases by categorizing input data patterns. A fuzzy classifier is a type of fuzzy logic system that uses classification labels as rule consequents, instead of output fuzzy terms from membership graphs. As a result, defuzzification is not required in classifiers since output crisp values do not exists in classification schemes. Final classification is derived from selecting the label with the highest probability rating.

Fuzzy classifiers have been used to detect a wide array of diseases. The system in [18] has eight different diagnostic classes of heart disease and accepts information on the dynamics of cardiovascular systems. Input variables including Spectral Entropy, Poincare plot geometry, Largest Lyapunov exponent and Detrended fluctuation are derived from external artificial neural networks. In [21], authors focused on detecting breast and lung cancer using a combination of fuzzy-based and rough-based logic.

System inputs for breast and lung cancer classifiers are taken from the Orange Machine Learning Data Mining database.

ECG signal data has become a popular application for fuzzy classification. In [19], researchers developed a classifier which focuses on ECG wave features such as P waves, QRS complexes and T waves to detect cardiovascular related arrhythmias. The duration, amplitude, period and slope of these waves are used as input variables for the system. Classification labels for arrhythmia include atrial tachycardia and ventricular tachycardia. In [23], engineers present another fuzzy-based classifier based on a single channel ECG lead. Various features that are evaluated include duration, beat position, interval, width and amplitude. There are more than 20 arrhythmias that are applied as labels in their implementation.

Others engineers have used fuzzy classifiers to measure the relevancy of medical information to a particular diagnosis. The objective of [24] is to use the fuzzy logic classification system to identify "relevant" ECG features. Researchers were able to breakdown the philosophy of conventional fuzzy logic and establish a method of measuring input relevance. Researchers based their technique on:

- The *output volume* of each output class in the system
- The *information gain* of each input feature based on the output volume
- The *normalized information gain* of each input relative to other inputs

These key measurements are essential to proving which ECG feature is relevant to a medical decision. Researchers are able to prove that some features are most important than others and achieve comparable classification results without those features with low importance. This paper

is important to understanding how individual input variables affect the dynamics of the output variable.

In [22], Saha and Chowdhury present a high performance FPGA-based processor for executing fuzzy operations. The architecture is based on the use of customized IP blocks to implement conventional style fuzzy logic that with a focus on high parallelization and low cost. The IP blocks include instances of detection units, antecedent units, rule units and defuzzifier units**.** Detection and Antecedent units are responsible for identifying which fuzzy terms are activated for given inputs with their corresponding degrees of truth. The rule unit assigns a firing strength for each rule consequent. The rule unit also solves for the numerator and denominator of the Yager-based defuzzification method. The defuzzifier unit performs the division operation to find the quotient output. These customized blocks provided extra memory savings and dynamic parameter settings but with greater design complexity and more difficult interpretability. Their system uses medical data like urea, glucose, creatinine, blood pressure levels to predict patient criticality.

Medical diagnosis is a very difficult procedure to automate using fuzzy engineering concepts. Many of the designs in this section require the knowledge of medical experts. Therefore, it is important to have a system that is both interpretable and trainable. In our patient risk controller, we design a system that expands on medical classification and introduces an output for disease risk. This will enable doctors to predict the current and future health of a patient based on their risk trajectory. In the design of our fuzzy controller, we must consider the following factors:

- Interpretability of rule matrix and fuzzy memberships
- Trainability of the controller with existing clinical medical datasets

- Nonlinear disposition of highly complex medical diagnosis
- Increased efficiency and accuracy without sacrificing speed or area
- Programmability of a design onto hardware/software platforms
- Application flexibility and reusability of the fuzzy logic design

When observing the points outlined above and considering the fuzzy models available, it was determined that the best model to fashion our system after was the genetically trained conventional model. This model maintains the interpretability required by doctors to monitor and ensures a systematic method for training and optimizing fuzzy parameters for medical diagnostic applications with complicated nonlinear target datasets. The more effective center of area defuzzification method can be used while preserving speed and area by parallelizing and consolidating the architecture implemented on an FPGA-based platform. Our fuzzy logic hardware design seeks to combine the best advantages and eliminate the worst disadvantages from each of the related papers presented in this chapter while achieving the design targets outlined above. The next section will explain how genetic algorithms have been developed over the years.

3.3 – GENETIC OPTIMIZATION DESIGN

Over the years, many researchers have designed various models for genetic algorithms. This section will present several papers which focus on the realization of genetic algorithms for a wide array of applications. Three different categories of algorithms are discussed in this section: (1) software-

based algorithms, (2) hardware-based algorithms and (3) hybrid systems with both hardware and software components. Hardware based architectures increase efficiency and decrease execution time while software based architectures provide simple implementation of complex applications.

In [25, 28], researchers designed a FPGA-based system for genetic algorithms written in VHDL. Their architectures contain a coarse-grained pipeline of customized genetic operator modules. These hardware modules implement genetic processes like fitness testing, selection, crossover and mutation. Both papers employ roulette wheel selection but use different techniques for crossover and mutation. Extra hardware blocks are included to store population candidates and generate random numbers. Their research explores the effectiveness of different combinations of evolutionary techniques for both linear and exponential functions for 4-bit, 8-bit and 16-bit based genetic algorithms. Both papers demonstrate the functional success of the system with comparable improvement to software-based version.

Parallelization is a major aspect of hardware based algorithms that make it very attractive. Researchers in [25] explored various ways to improve execution time by creating parallel genetic operator modules. To alleviate a major bottleneck in the execution time, two selection modules are parallelized to produce two parents at a time. In [26], Tang and Yip design a FPGA-based system for genetic algorithms on a PCI board. They accomplish parallelization by integrating multiple boards to available PCI slots on a computer. Results exhibited a significant speedup over the software-based genetic algorithm.

Genetic algorithms offer major advantages in searching for solutions in many different applications, however, some applications are too

complicated to implement in fitness hardware modules. In [25], fitness functions are designed in software and converted to hardware using a PRISM-1 translator. Authors of [27] developed a hybrid version of the algorithm by integrating a software fitness function into the hardware architecture. These software provisions increase the capability of the hardware-based algorithm to handle complicated systems.

Another area where software applications help genetic algorithms is with parameter modifications. Genetic algorithms often require that specific parameters are manually tuned to achieve the best results, which can be difficult with FPGA-based hardware designs. As a result, some papers include user interfaces in their designs enabling users to modify these algorithm parameters. In [25, 27], parameters like population size, individual length, crossover and mutation probabilities and other important specifications are modified via user interfaces. This type of software accommodation allows for user-defined changes to the algorithm specifications for increased effectiveness.

Binary-based genetic algorithms programmed on FPGA devices provide many advantages ranging from executive time to limited area and power costs. However, integer-based software algorithms offer increased design capability for complex applications like fuzzy logic, which are not easily replicated in a hardware-based fitness function module. The next section focuses on both forms of genetic algorithms that optimize fuzzy logic technology. Several options are introduced which illustrate how fuzzy logic can be systematically designed and perfected using genetic programming.

3.4 - EVOLUTIONARY ALGORITHMS FOR FUZZY LOGIC

Over the years, many papers have discussed how evolutionary algorithms optimize fuzzy logic controllers. This section will survey research that has been done to improve the effectiveness of designing fuzzy controllers. Even though the process for formulating the algorithm is similar, the encoding scheme, fitness function and other algorithm specifications differ between various controller applications.

The authors of [29] describe how genetic algorithms optimize various types of fuzzy models including conventional, clustering, neuro-fuzzy and TSK fuzzy control systems. Genetic algorithms can search for the best fuzzy memberships and set of rules for conventional models or the weights of neurons in the neuro-fuzzy model. This section will focus on papers that focus on designing conventional fuzzy models.

Encoding the fuzzy sets and rule matrix is a fundamental requirement for genetic algorithms. In [31] and [32], researchers have tried various methods to encode these fuzzy parameters. The fuzzy membership graphs are designed using triangular fuzzy membership graphs. In [31], the fuzzy membership graphs are encoded by storing the location of the left, center and right points of each fuzzy set triangle. This encoding scheme allows for simple and effective translation but requires a fifteen gene chromosome to encode a fuzzy membership graph with five fuzzy sets. To limit the complexity of the design, researchers in [32] designed a formula for calculating these same points using only nine position points.

In [35], the fuzzy membership graphs are designed using the Gaussian function, requiring two variables per function. A ten gene chromosome encodes the fuzzy membership graph is their algorithm design. In all three papers, the rule matrix follows a relatively simple encoding format. The position of each consequent in a matrix corresponds to the antecedent of each rule proposition and can directly translate into a position in the rule chromosome. Therefore a rule chromosome only requires the consequents which shortens its size and simplifies the search space.

Genetic algorithms can also eliminate unnecessary fuzzy sets or rule propositions by adding control genes to the encoded chromosome. Decreasing the number of fuzzy sets and rules can reduce complexity. Yang et al added control genes to the fuzzy membership chromosome to analyze which fuzzy sets could be repealed [31]. Castro and Carmargo inserted control genes to the rule chromosome to determine which rules are unnecessary [30]. This adds to the complexity of the algorithm but ensures limited fuzzy logic design complexity.

Fitness functions vary based on the control applications being developed. For a traffic control system, fitness formulas are designed as a function of the average vehicle delay or total vehicle delay [31, 32]. A classification system uses classification success rates as the primary variable to measure fitness [30, 35]. Other applications utilize approximation formulas to measure the fitness of a controller based on predicted training data sets. In [33], Chiou et al uses the mean squared error formula to determine the fitness of solution candidates. The ability to compare the measured output of the controller to predicted output of the training set offers an effective approximation-based fitness scheme that scores each solution candidate.

The next step of algorithm implementation is the creation of genetic operator specifications. Many papers offer standard population sizes, selection, crossover and mutation techniques and probabilities to guarantee algorithm success. However, researchers in [30] introduced a genetic diversity meter which measures how diverse the population is in an effort to avoid premature convergence. This permits the algorithm to dynamically adjust genetic operator probabilities ensuring search effectiveness for globally optimum solutions.

When genetically optimizing fuzzy logic parameters, a chromosome can include control genes, encoded fuzzy membership graphs and encoded rule matrices. Elongated chromosomes sometimes have negative effects on the success of the search algorithm. As a result, many engineers have developed several methods for resolving this issue.

One method that has been introduced is the creation of successive genetic algorithms that optimize fuzzy parameters separately [30]. To develop fuzzy rules for their controller, they used one genetic algorithm to find the optimal set of rules using an encoded chromosome to represent a rule matrix. Another genetic algorithm is then executed to find the most effective and smallest combination of those rules using an encoded chromosome to represent the rule matrix control genes. This successive algorithm allows engineers to find an optimal set of limited rules that can most effectively fit the fuzzy controller.

Another popular method is called the iterative algorithm which seeks to optimize fuzzy parameters using a two layered system [32, 33]. The objective is to find solutions for fuzzy parameters using shorter encoded chromosomes which can improve the process. The two layer system contains a top layer which optimizes the fuzzy membership graph and the

bottom layer which optimizes the rule matrix. The top layer runs a genetic algorithm which searches for the best fuzzy membership graphs using fuzzy rules retrieved from the bottom layer. The bottom layer runs a genetic algorithm which searches for the best fuzzy rules using fuzzy membership graphs retrieved from the top layer. This process occurs back to back between layers in sequence until a final stopping condition is satisfied. Iterative algorithms allow for multi-layered genetic processes to select a combination of optimized solutions for a single system. The steps outlined below demonstrate how iterative algorithms flow.

[1] Tuning Rule Propositions
  1.1. Initialize the population of encoded rule matrix chromosomes
  1.2. Evaluate initial population using membership graphs learned from [2]
  1.3. Begin reproductive cycle
    1.3.1. Perform competitive selection
    1.3.2. Apply genetic operators to generate new solutions
    1.3.3. Evaluate solutions in the population
  1.4. Repeat step [1] until some convergence criteria is satisfied

[2] Tuning Membership Graph
  2.1. Initialize the population of encoded membership graph chromosomes
  2.2. Evaluate initial population using rules learned from [1]
  2.3. Begin reproductive cycle
    2.3.1. Perform competitive selection
    2.3.2. Apply genetic operators to generate new solutions
    2.3.3. Evaluate solutions in the population
  2.4. Repeat step [2] until some convergence criteria is satisfied

[3] Repeat step [1] until the final fitness values from [2] levels off

In the very beginning of the iterative algorithm, a random membership graph solution is provided to step [1] since step [2] has not yet been executed.

Researchers have designed hardware FPGA-based systems that conduct optimization techniques online. Many algorithms are executed on separate systems, however, Cao et al developed a real-time context switchable fuzzy inference that could be optimized and updated online [34]. The chip architecture is separate into two parts: genetic software written onto a microcontroller and a reconfigurable fuzzy controller on an FPGA device. The FPGA hardware is designed using custom blocks to implement the conventional model of fuzzy logic. Configuration interface blocks allow for fuzzy parameter rules to be switched based on genetic software solutions. Some aspects of this architecture can be very useful in developing a generic fuzzy logic controller that can programmed for many different applications. A major issue is the replacement of the traditional fuzzifier-rule matrix format for a more complex rule matrix network which directly accepts crisp input data as opposed to fuzzified labels. Although memory is saved from excluding the fuzzifier, added memory is required for the rule inference phase. Also, the process of switching and updating the rule matrix becomes more complex and inconvenient when the number of inputs or the number of input data bits increases.

This chapter describes how evolutionary algorithms optimize and improve the development of fuzzy logic controllers for various applications like medical diagnosis. Traditionally, fuzzy classifiers have often been used to identify the existence of diseases but they fail to provide information that promotes prevention. The goal of this research is to design a controller which enables medical professionals to track and record the trends of diseases in patients. This is accomplished by training controllers to detect the patient's risk of having these diseases based on existing clinical trials and medical surveys. The next chapter focuses on the basic architecture of

our genetically tuned fuzzy controller and outlines various contributions and improvements made over existing research models.

4.0 - HARDWARE-SOFTWARE SYSTEM

DESIGN

4.1 – DESIGN EXECUTION FLOW

Our proposed design is separated into two sub-systems: fuzzy logic controller and genetic optimization program. The fuzzy logic controller performs the fuzzy operation and is designed on a FPGA-based hardware platform. The genetic optimization program is written in software and executed on a PC. The figure below describes a rough overview of the design.

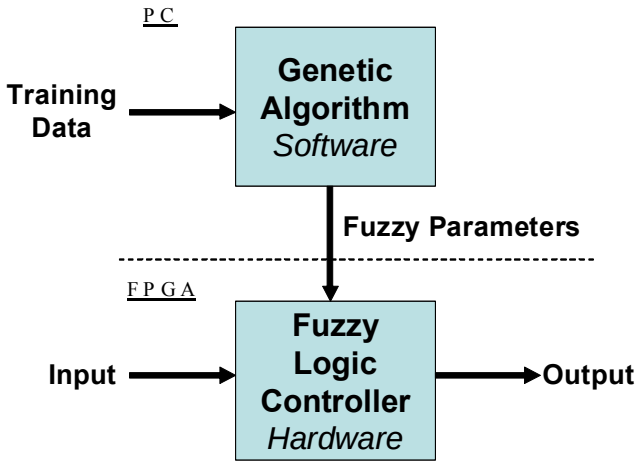

Figure 4.1 – System Design Overview

The fuzzy logic controller operates in two modes: optimization and execution. During optimization, memory contents of the controller are manipulated using genetically optimized fuzzy parameters. These parameters are converted into usable fuzzy membership graphs and rule propositions and are stored as internal memory. During execution, fuzzy logic operations are triggered via an external hardware interrupt. This trigger initiates a series of successive triggers that occur between functional stages in the fuzzy controller. System inputs are entered by the user into the controller which produces an output value.

The genetic program performs the optimization algorithm which selects the best fuzzy parameters for a given set of training data. Training data is provided by the user derived from expert evidence based research. This technique provides the controller with encoded sets of optimized fuzzy membership graphs and rule propositions which are stored in fuzzy context registers. In our model, we used an improvised version of iterative evolutionary algorithms due to the complexity of the biomedical application.

Since a very large solution space exists, modified versions of existing iterative algorithms and multiple sets of training data are used to help find the best parameters.

4.2 – FUZZY LOGIC HARDWARE

To implement our fuzzy logic controller, we reference the conventional modeling technique for several reasons. Conventional models are very interpretable and flexible which allow for simpler reconfiguration and manipulation. These attributes make conventional models more attractive than TSK modeling since definitive rule propositions are used to link the input and output fuzzy sets. Even though TSK models have more stable output responses for real-time systems, our device is intended to operate in a non-real time environment.

Another important aspect of conventional-based fuzzy systems is the ability to design the controller on a FPGA device. The utilization of limited customized hardware blocks makes the implementation of conventional fuzzy models more desirable than neuro-fuzzy models. FPGA-based solutions for large artificial neural networks have presented many challenges to researchers due to the large resources needed to implement the neural net [39]. The next section will provide an overview of the hardware architecture of the system based on conventional modeling.

As stated earlier, our design will include many of the advantages outlined in related papers. The goal is to maintain speed and resource

advantages while involving more effective defuzzification methods. Application flexibility and context switchability are also vital aspects of the design that must be preserved. The following sections will discuss in detail the architecture of the design being proposed.

4.2.1 - Hardware Design Overview

Our fuzzy logic system is based on a combination of special features drawn from several research papers discussed in previous sections of this paper. The system is parallelized and separated into six stages: fuzzify, rule, minimum, multiplex, output and divide. The diagram below provided a block diagram of the modules used to implement the system.

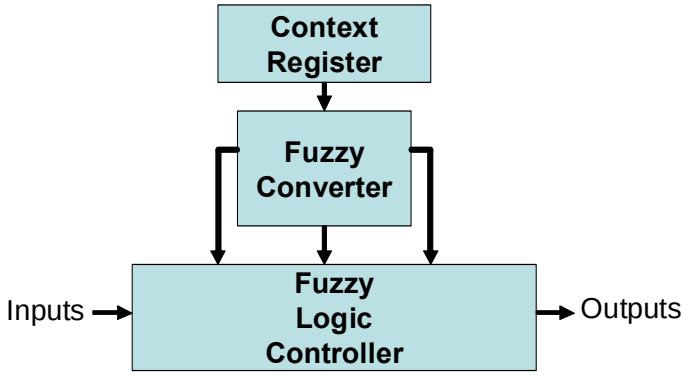

Figure 4.2 – Hardware Design Overview

Context registers and conversion modules store encoded fuzzy parameters and translate them into interpretable fuzzy information. Memory blocks are used to store this information, which include fuzzy membership graphs and rule matrix propositions. These modules enable the fuzzy logic controller

block to be reconfigured by simply modifying the contents of the context registers. The next section will discuss some design considerations that were applied to our hardware model.

4.2.2 - Design Considerations

This section lists several points that must be considered concerning the physical application when designing our system. The controller we are designing is not expected to be a real-time system. Therefore, a trigger signal will be used to prompt the start of the fuzzy algorithm. Another trigger signal is needed for fuzzy reconfiguration based on parameters recorded in the context registers. Input and output user interfaces are also required to send medical data and receive risk percentages. Binary dipswitch, 7-segment LED and pushbutton embedded devices are linked to the Xilinx ML509 Evaluation Board FPGA evaluation board.

The speed of the hardware device is mildly important, but more priority will be devoted to the functionality and accuracy of the operation. A powerful FPGA device with significant logic and memory sub-components can increase the efficiency and flexibility of the controller. Therefore, the latest Xilinx Virtex-5 XC5VLX-110T FPGA device will be employed in our design.

To accurately represent the fuzzy logic controller in binary form, the following system specifications are listed below:

- Input/Output Crisp Data – 8-bit resolution
- Membership Degree – 6-bit resolution
- Fuzzy Set Label – 3-bit resolution

- Maximum Fuzzy Graph Overlap – 2

This section briefly outlined some details of the design that were taken into consideration. The next section will discuss the architecture of the hardware-based fuzzy controller.

4.2.3 - Controller Architecture

The controller architecture is organized according to three fuzzy logic algorithm phases: fuzzification, inference and defuzzification. The phases are implemented in five stages: fuzzify, rule and minimum, multiplex, output and divide. The following block diagram shows the modules used to implement each phase.

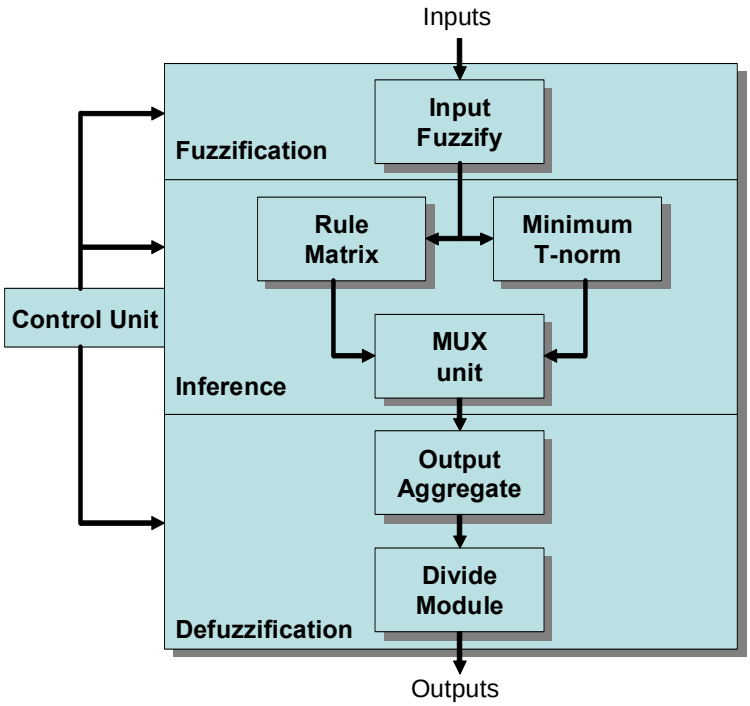

Figure 4.3 – Controller Architecture

The first three stages in the architecture are fully parallelized and each require one clock cycle. The final two stages require several cycles to complete the defuzzification process. The number of each unit used in the structural design depends on the model specifications of the fuzzy controller. The number of inputs, level of overlap and the number of fuzzy sets are important factors that affect the size and organization of the design. Our fuzzy controller specifications include:

- 1-3 inputs
- 1 output
- 2-5 fuzzy sets
- triangular fuzzy set functions
- 64 membership levels
- 256-element universe of discourse
- overlap of 2

4.2.3.1 - Control Unit The control unit is a finite state machine that allows the system to transition between three states: reset, optimization and execution. The control unit module inputs user triggers and outputs mode and enable signals to control the operation and activation of each fuzzy hardware module. During configuration mode, the control unit activates the input fuzzify unit, rules matrix unit and output aggregate unit and switches their mode to a write operation. The control unit also transmits memory addresses to the fuzzy converter. This address is translated into membership graph and rule information which is stored in its respective memory location.

After optimization is completed, the control unit shifts to the execution mode. During execution mode, the control unit switches the memory blocks to read mode and also activates the remaining hardware modules. The control unit plays a limited role in the execution mode since successive triggering methods allow each stage to be triggered by its previous stage. This concept is derived from the asynchronous fuzzy controller designed in [16]. The diagram below displays how the state machine is set up.

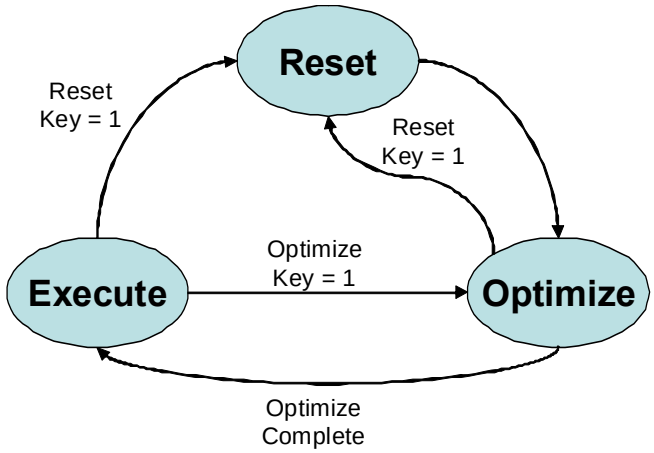

Figure 4.4 –Control Unit State Machine

The next figure is a block diagram of the control unit. The fuzzy unit enables and modes represent the signals that are sent to the input fuzzify, rules matrix and output aggregate units. Membership and Rule addresses represent data that is sent to the fuzzy converter modules and internal memory.

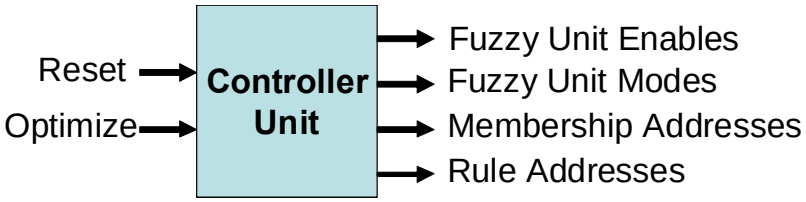

Figure 4.5 – Control Unit

The signals in the block diagram area:

- Reset – 1-bit
- Optimize – 1-bit
- Fuzzy Unit Enable Bus – 4-bits
- Fuzzy Unit Modes Bus – 4-bits
- Membership Addresses – 8-bits
- Rule Addresses – 8-bits

4.2.3.2 - Fuzzy Membership and Rule Converter Although these two modules appear outside the controller architecture, they are critical components to the overall design. They are responsible for translating and transferring fuzzy context data to the memory blocks within the architecture. Any component that stores and load fuzzy membership graph data and fuzzy rules must be linked to the fuzzy converters. Each converter is directly initialized and managed by the control unit. Context register provides the encoded information needed to accurately.

Fuzzy membership converters accept input 8-bit address signals from the control unit and calculate their respective fuzzy labels and degree of memberships. The converter constructs and transfers the membership graphs to the input fuzzify unit and output aggregate module. Their memory

blocks store membership graphs for both input and output variables which are used during the fuzzy logic operation. A fuzzy membership converter is required for each input and output variable included in the fuzzy controller architecture. The method used for calculating the membership graph plots is similar to the binary search algorithm developed in [22]. Triangular shapes are used to represent fuzzy membership functions. The figure below illustrates how the binary search algorithm uses the boundaries of rising and falling sections of the fuzzy graphs to begin the search. These boundaries are selected using the genetic training algorithm discussed in the next major section.

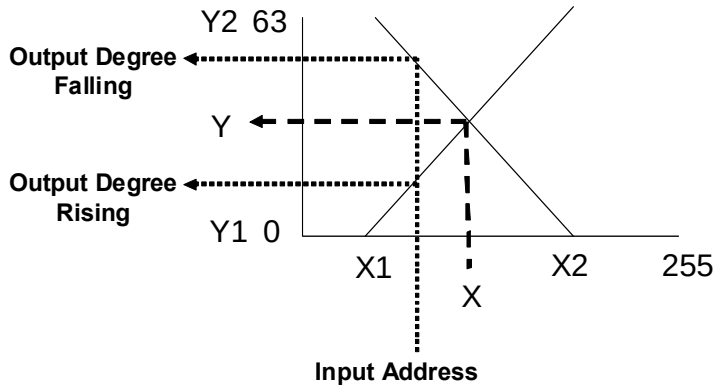

Figure 4.6 – Triangular Fuzzy Membership Graphs

The following algorithm searches for the output Y (membership degrees) using the input X (controller address signals) for the rising sections of fuzzy graphs.

[1] X = (X2 – X1) / 2 + X1;

[2] Y = (Y2 – Y1) / 2 + Y1;

[3] If X > Input Address Then Output Degree is between Y1 and Y else skip to [4]

3.1. X2 = X && Y2 = Y;

3.2. Repeat [1];

[4] If X < Input Address Then Output Degree is between Y and Y2 else skip to [5]

4.1. X1 = X && Y1 = Y;

4.2. Repeat [1];

[5] If X = Input Address Then Output Degree is Y

5.1. Output Degree = Y;

5.2. End search;

The following algorithm searches for the output Y (membership degrees) based on the input X (controller address signals) for the falling sections of fuzzy graphs.

[1] X = (X2 – X1) / 2 + X1;

[2] Y = (Y2 – Y1) / 2 + Y1;

[3] If X > Input Address Then Output Degree is between Y and Y2 else skip to [4]

3.1. X2 = X && Y2 = Y;

3.2. Repeat [1];

[4] If X < Input Address Then Output Degree is between Y1 and Y else skip to [5]

4.1. X1 = X && Y1 = Y;

4.2. Repeat [1];

[5] If X = Input Address Then Output Degree is Y

5.1. Output Degree = Y;

5.2. End search;

These two algorithms provide effective and efficient means for calculating the graphical plots. The maximum number of cycles needed to search for the output is 8. The converter requires knowledge of the number of fuzzy sets for that particular membership graph.

Fuzzy rule converters use a simpler method to store and transfer data. Data from the context registers are carefully organized to ensure easy translation. The encoded rule matrix in the fuzzy context register is

structured in a pre-arranged order that is identical to the arrangement in the memory module. The control unit sends a 9-bit address containing three 3-bit fuzzy label signals which enables the converter to decode the location of a particular rule proposition and transfers the derived rule consequent to the same location in the rule memory module. Both converters are enabled and triggered simultaneously during the configuration phase of the converter. A total of 256 cycles are required to complete the fuzzy membership converter phase. The fuzzy rule converter requires 127 cycles to completely decode and transfer the rule matrix. The following two diagrams display the signals of our converter modules.

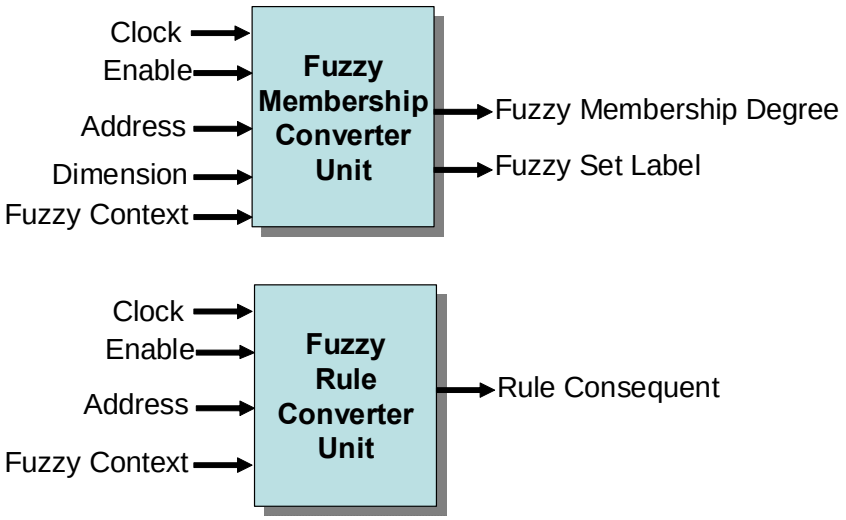

Figure 4.7 – Fuzzy Converter blocks

- Clock, Enable – 1 bit
- Membership Address – 8-bit
- Dimension – 3-bit (number of fuzzy sets)

- Rule Address – 9-bit
- Rule Consequent – 3-bit
- Fuzzy Context – varies depending on size of genetically tuned parameters
- Fuzzy Membership Degree/Fuzzy Label – 6-bit/3-bit

4.2.3.3 - Input Fuzzifier The input fuzzifier unit is a memory block which stores the data for a single input fuzzy membership graph. A single port RAM contains the fuzzy set label and membership degree which is addressed by the crisp input data received from the user. The setup of the memory arrays is similar to the optimized architecture in [12] in which fuzzy set labels and their corresponding degrees of truth share two memory arrays. The universe of discourse for the input contains 256 elements and 64 membership levels. The system is also designed to handle up to 5 fuzzy sets for each input. Each line in memory has the fuzzy set label and membership degree. A total of 9-bits are used on each data line (3-bit label + 6-bit degree). The memory unit has 256 9-bit data lines. Read and write signals allow for easy access to memory content retrieval and modification. During optimization phase, the write mode is enabled and the address bus is controlled by the control unit. Membership graph data is received from the fuzzy converter unit. During the execution phase, the read mode is enabled and the address bus is controlled by the system input. Figure 4.8 displays the input fuzzier unit.

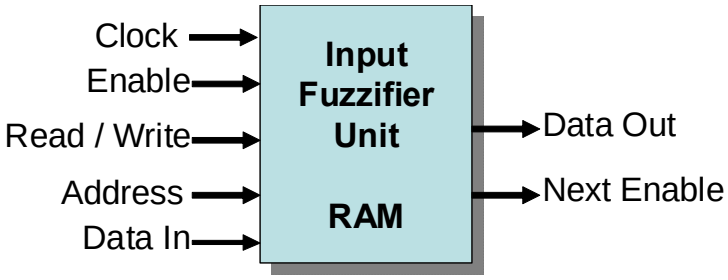

Figure 4.8 – Input Fuzzify Unit

Our design contains three input fuzzy RAMs for our three inputs and requires a single clock cycle for in read mode. The RAM block signals are:

- Clock, Enable, Next Enable – 1-bit each
- Read / Write – 1-bit
- Address – 8-bits (input data)
- Data In – 9-bits (3-bit fuzzy label + 6-bit membership degree)
- Data Out – 9-bits (3-bit fuzzy label + 6-bit membership degree)

4.2.3.4 - Rule Matrix The rule matrix is implemented in a RAM memory block. Antecedent fuzzy labels are transferred from the input fuzzy unit. These inputs are translated into RAM memory addresses which locate rule consequents stored in memory. Memory is loaded with 3-bit fuzzy term labels from the output membership graphs. The size of the memory is equal to the number of total possible rule propositions in inference process. In this particular design, we can contain up to 125 rules therefore the memory unit is designed to have 127 3-bit data lines. The following diagram shows how the memory is arranged for a two input 5x5 rule matrix with 25 rules.

Rule Matrix

	A	B	C	D	E
V	1	2	3	4	5
W	6	7	8	9	10
X	11	12	13	14	15
Y	16	17	18	19	20
Z	21	22	23	24	25

Rule Memory

1
2
3
4
:
23
24
25

Figure 4.9 – Rule Matrix and Memory Location synchronization

A three dimensional matrix would continue the count sequence into successive pages. Any unused rule proposition in the 127 element rule memory is set to 0 in the memory block. To calculate the memory location from activated input fuzzy set labels during a fuzzy operation, we implement the following formula:

Rule Memory Address = (*label 1* – 1) * 5 + (*label 2* – 1) * 1 + (*label 3* – 1) * 25

We also include some fundamental conditions to eliminate operational hazards and other run-time errors. The dimension, or number of fuzzy sets, in each system input is used as conditional variables. If a system input is inactive the dimension is set to (1), since each input requires at least (2) fuzzy sets. Block RAMs can be used to implement the memory unit. Read and write signals are used to switch the memory to store or load rule data. The figure below shows the rule matrix block diagram.

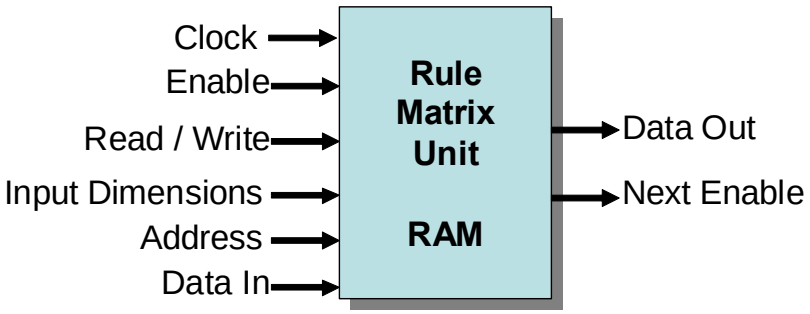

Figure 4.10 – Rule Matrix Unit

To optimize the design, the number of the rule matrix blocks is limited to the number of rules that are active during a single cycle [12]. Using 8 rule matrix units promotes parallelization while limiting costs in our design by decreasing the required clock cycles to one in write mode.

No. of rules activated = (No. of inputs) ^ overlap

The RAM block signals are:

- Clock, Enable, Next Enable – 1-bit each
- Read / Write – 1 bit
- Input Dimensions – 9-bit (No. of fuzzy sets for each input)
- Address – 8 bits (input data)
- Data In – 3 bits (fuzzy label)
- Data Out – 3 bits (fuzzy label)

4.2.3.5 - Minimum T-norm The minimum T-norm unit is responsible for determining the firing strength of each active rule proposition. Each minimum T-norm unit receives the membership degrees from the input fuzzify units. The module outputs the minimum truth value for the fuzzy set

labels of activated rules. The following block diagram shows the T-norm unit.

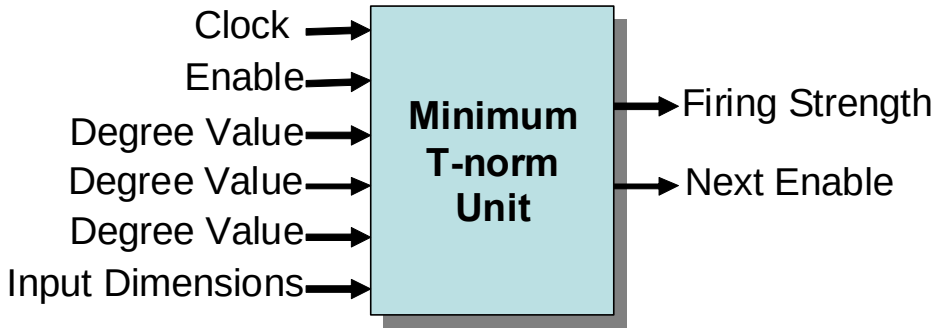

Figure 4.11 – Minimum T-norm Unit

The number of T-norm units is equal to the number of rule matrix units. Our optimized design includes 8 modules. This phase requires only one clock cycle. The block signals are:

- Clock, Enable, Next Enable – 1-bit each
- Degree Value – 6-bits
- Firing Strength – 6-bits

4.2.3.6 - Multiplexer (MUX) The MUX unit determines the truth value of the output fuzzy set labels. The module performs the max operation on active firing strengths of corresponding consequent fuzzy labels [12]. This operation is necessary for the next process of aggregating the final output graph. The MUX unit accepts the firing strength and consequents of the active rule propositions from the minimum T-norm and rule matrix blocks. It also outputs the truth value of each output fuzzy label respectively. In our design, the unit accepts 8 consequent and their firing strength pairs and

sends 5 degrees of truth, matching the 5 fuzzy sets of our output membership graph.

For M Consequent & Firing Strength Pairs

If consequent = Output label N

Output N degree of truth = MAX(Current Max, Firing Strength)

End

End

The module uses only one clock cycle. A block diagram is shown below.

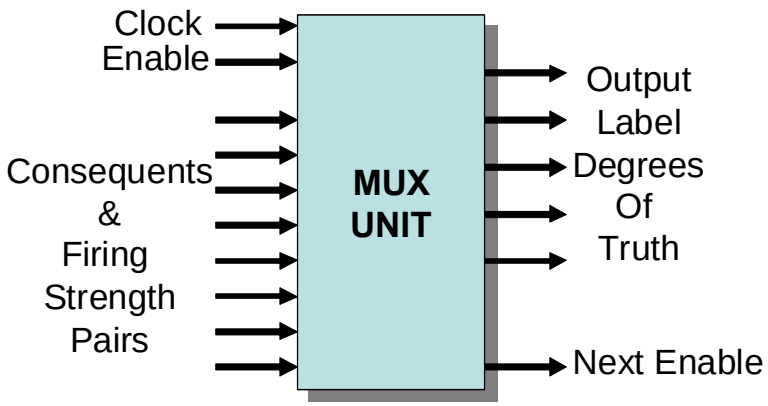

Figure 4.12 – MUX Unit

The signals on the block are:

- Clock, Enable, Next Enable – 1-bit each
- Consequents – 3-bits
- Firing Strength – 6-bits
- Output Degrees – 6-bits

4.2.3.7 - Output Aggregate The output aggregate unit determines the mass and area of the final aggregate graph. In many fuzzy controller designs, aggregate graphs are not necessary since center of gravity provides simpler computations. However, our design requires greater efficiency so as a result the center of area method is implemented. This method requires that the mass and area of the final aggregate output graph are calculated.

This aggregate graph is formed by using the maximum T-norm operation on output fuzzy sets which are manipulated based on the degrees of truth supplied by the MUX unit. This process is described in Section 2.1.2. A single-port block RAM stores the pre-set output membership graph containing 256 elements over across the universe of discourse and is used as a reference to form the aggregate graph. To decrease execution time, both area and mass are calculated simultaneously. The function below describes how we calculate area by summing the degrees of truth under the aggregate graph.

$$\textbf{Area} = \int \textit{f-aggregate}\,(J),\quad 0 < J < 256$$
$$= \sum \textit{f-aggregate}\,(J),\ 0 < J < 256$$

The following function describes how we calculate mass by summing the volume under the aggregate graph.

$$\textbf{Mass} = \int J * \textit{f-aggregate}\,(J),\quad 0 < J < 256$$
$$= \sum J * \textit{f-aggregate}\,(J),\ 0 < J < 256$$

The accumulation of mass and area is highly dependent on the truth level of the aggregate graph at each discourse element, ***J***. A combination of Min and Max fuzzy operations are employed to determine each truth level based on

the output fuzzy sets and the output degrees from the MUX unit where the overlap factor is at most 2. The following functions describe the fuzzy operation.

$$\textbf{\textit{Min\_1}} = \textbf{MIN (} \textbf{\textit{f-degree}} \textbf{ (J), Output Degree (} \textbf{\textit{f-label}} \textbf{ (J)), 0 < J < 256}$$

$$\textbf{\textit{Min\_2}} = \textbf{MIN (} \textbf{\textit{f-degree*}} \textbf{ (J), Output Degree (} \textbf{\textit{f-label*}} \textbf{ (J)), 0 < J < 256}$$

$$\textbf{\textit{f-aggregate}} \textbf{ (J) = MAX (Min\_1, Min\_2), 0 < J < 256}$$

$$\textbf{* = 2}^{\textbf{nd}} \textbf{ Overlap Layer (Degree \& Label for unused non-overlapping region is 0)}$$

These three functions allow us to complete the manipulation process described in Figures 2.18- 2.20. And since both the aggregate graph and mass and area accumulation are sequential process that can be directly linked, both processes were combined into one sequential function where the calculated aggregate truth level was directly added to the mass and area accumulation function. This combination led to further timing savings.

In both optimization and execution mode, a single output aggregate unit requires 256 cycles to determine the mass and area of the center of area function. However by increasing the number of output aggregate units, exponentially higher parallelization and decreased execution time can be achieved. By distributing the storage of a single output membership graph to four aggregate units evenly, execution time could be cut by 75%. For example, our single module stores 256 elements requiring 256 cycles to process. If four modules were employed to store 64 elements, it would only require 64 cycles to determine the mass and area of each region. The total mass and area could simply be the sum of four values. The figure below shows an example the hardware module.

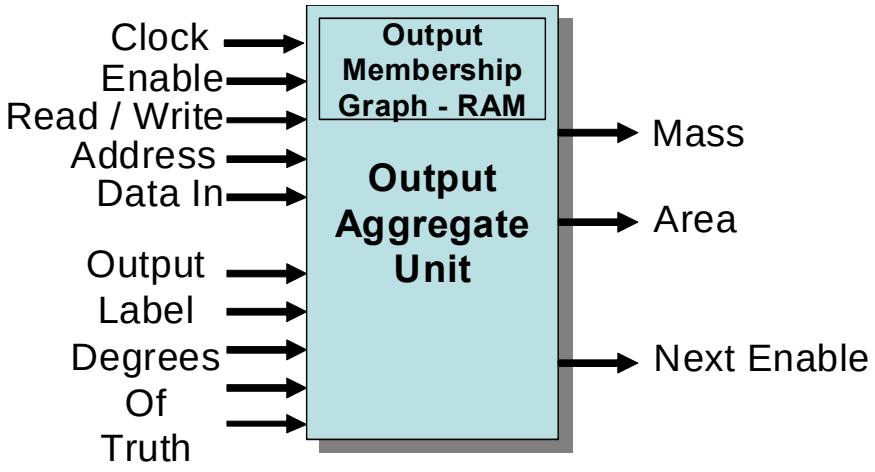

Figure 4.13 – Output Aggregate Unit

The block signals are:

- Clock, Enable, Next Enable – 1-bit each
- Read / Write – 1-bit
- Address – 8-bits (input data)
- Data In – 9-bits (3-bit fuzzy label + 6-bit membership degree)
- Output Degrees – 6-bits
- Mass – 24-bits
- Area – 16-bits

4.2.3.8 - Division Module The division module unit uses the mass and area data to solve the final output. There are many different hardware division methods that are available promoting fast and cost-effective division. To accomplish this goal, some hardware-based fuzzy controller models pre-store every possible quotient output in memory. Other models design their technique based on the storage of divisor reciprocals, enabling division to be performed through reciprocal multiplication. Although these

techniques offer great advantages, a 24-bit and 16-bit division operation would be too difficult to store. In our research we used two techniques: accumulator-based division and partial reciprocal-based division. The accumulator-based method requires limited hardware and is highly accurate with a long execution time which in our design is at most 256 cycles.

Accumulator Method

1. A = A + divisor
2. If A < dividend:
    - Increase Count
    - Repeat Step 1
3. Else Quotient = Count

Partial reciprocal-based methods allow for repeated division operations to occur using a limited set of divisor reciprocals.

Partial Reciprocal Method

1. M = base-2 log of (Maximum Value in reciprocal memory + 1)
2. N = no. of bits in each reciprocal
3. If bits [M-1:0] >= 2
    - Multiply dividend and divisor by the reciprocal of bits [M-1:0]
    - If dividend (N-1) = 1: dividend_round_up = true; Else false;
    - If divisor (N-1) = 1: divisor_round_up = true; Else false;
    - Right shift dividend and divisor N places
    - Add +1 to dividend or divisor if respective round up = true
4. Else Right shift dividend and divisor M places
5. If Divisor > 1: Repeat Step 1
6. Else Quotient = Dividend

For our system, we store the 16-bit reciprocal for any value up to 16 besides 0 and 1, therefore M = 4 and N = 16. Our module requires at most 5 clock

cycles to determine output quotients. This method can significantly decrease execution times while using minimal additional memory and control in a cost-effective manner; however, accuracy of the output can be off by up to 5%. The figure below displays the hardware block. The output is the 8-bits.

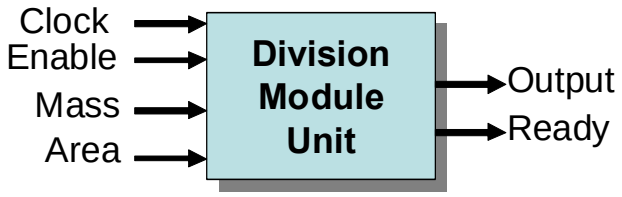

Figure 4.14 – Division Module Unit

The signals of the block are:

- Clock, Enable, Ready – 1-bit each
- Mass – 24-bits
- Area – 16-bits
- Output – 8-bits

4.2.4 - VHDL Implementation

Our design is written using VHDL programming language and implemented on a Xilinx Virtex-5 FPGA device. This section will outline specific sections of VHDL code that implements the architecture described in the previous section. The code is written using a similar hierarchy displayed in the architecture. The top file of the fuzzy model is the ***fuzzy.v***

file. This file contains all of the controller sub-components and their interconnected signals including the fuzzy converter modules.

The outline below briefly describes the main tasks and responsibilities of each VHDL program.

- ***controller.v***
  - controls the mode and activation: ***fuzzify.v***, ***rules.v***, ***output.v***
  - sends memory addresses to fuzzy parameters converters
- ***variables.v***
  - stores input fuzzy membership graph information
  - determines input fuzzy labels and degrees of truth
- ***fuzzy_conv.v***
  - converts fuzzy context to fuzzy membership graph information
- ***rules.v***
  - stores fuzzy rule proposition information
  - determines fuzzy consequents of rule propositions
- ***rules_conv.v***
  - converts fuzzy context to fuzzy rule proposition information
- ***min.v***
  - calculates the firing strength of each rule proposition
- ***mux.v***
  - assigns degrees of truth to output fuzzy labels
- ***output.v***
  - stores output fuzzy membership graph information
  - computes aggregate graph mass and area
- ***divide.v***
  - divide mass and area to retrieve center of area output value

These VHDL files are compiled and synthesized to form the following system architecture shown in following figures.

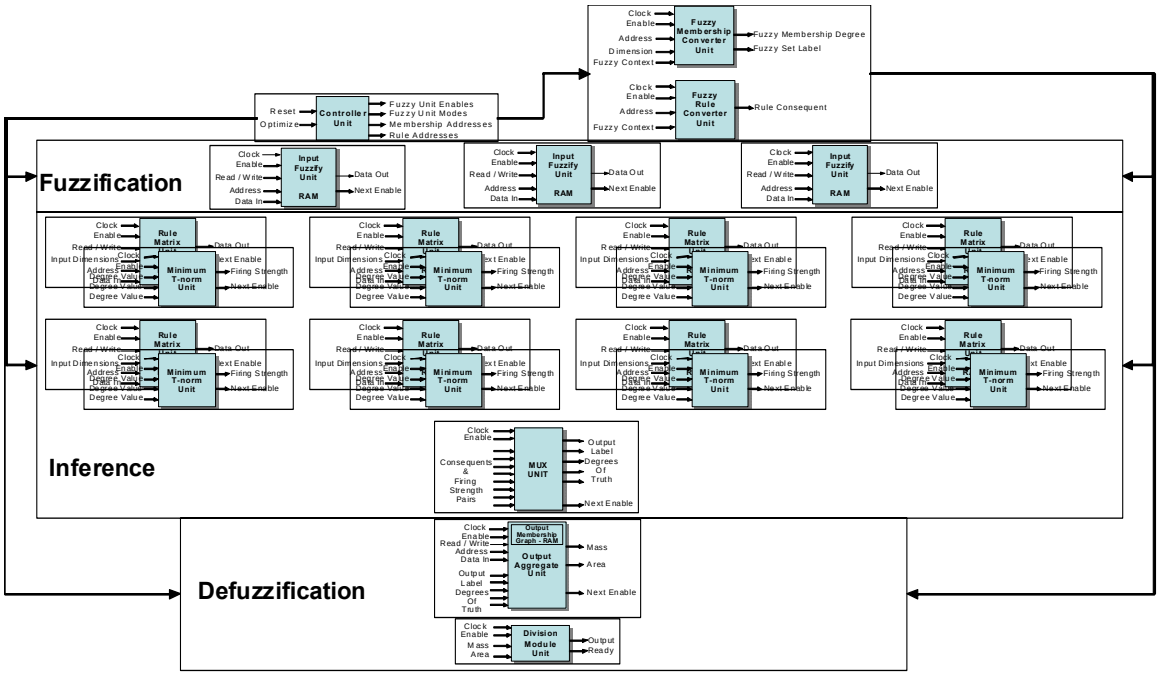

Figure 4.15 (a) – Top-Level Modular Hardware Schematic

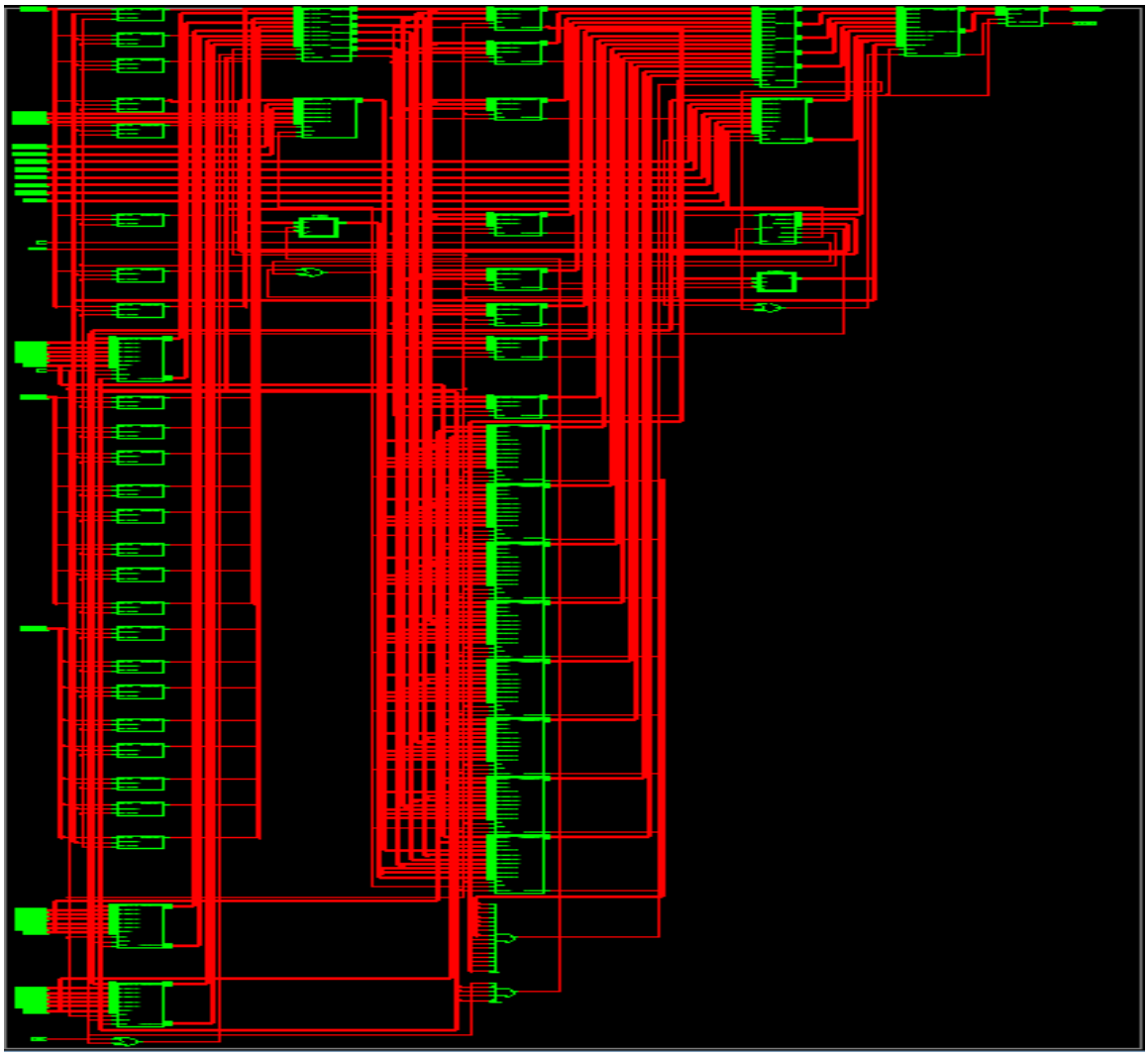

Figure 4.15 (b) – RTL Top-Level Hardware Schematic

The input and outputs of the system is outlined below. Besides the system data inputs, the hardware block also contains designated input ports for external genetic context registers.

- Data Input1
    - 8 bit input for input 1
- Chromosome1
    - 40 bit encoded chromosome for fuzzy membership of input 1

- Dimension1
    - 3 bit number of fuzzy sets in input1
- Data Input2
    - 8 bit input for input 2
- Chromosome2
    - 40 bit encoded chromosome for fuzzy membership of input 2
- Dimension2
    - 3 bit number of fuzzy sets in input 2
- Data Input3
    - 8 bit input for input 3
- Chromosome3
    - 40 bit encoded chromosome for fuzzy membership of input 3
- Dimension3
    - 3 bit number of fuzzy sets in input 3
- Rule Chromosome
    - 125 3-bit encoded chromosome for fuzzy rule matrix
- Data Outputs
    - 8 bit system output
- Chromosome4
    - 40 bit encoded chromosome for fuzzy membership of output

4.3 – EVOLUTIONARY ALGORITHM SOFTWARE

This section will focus on the implementation of our evolutionary algorithm which enables us to train our hardware-based fuzzy logic controller. Our software will employ the iterative algorithmic techniques to search for optimal fuzzy parameters necessary to effectively operate a patient risk evaluator. Since our genetic solution is multi-objective, using iterative algorithms can help us cater operator parameters to individual sub-solution algorithms. This allows for more efficient searches of a large solution space.

Measuring risk in patients is a very bold and ambitious application to implement. Training will require evidence based research surveys and trials to properly design the risk evaluator. Our controller would require expert knowledge guided by specific medical surveys that have a wide and complete range of proven samples. For a design with 3 inputs, 256-width universe of discourse and 64 membership levels, there are approximately 10^8 output data points. A large number of sample data points would be needed to ensure substantive genetic training. However, since many doctors do not directly evaluate patient risk, this particular type of information is not widely available on a large scale and presents a major challenge for our training.

Another challenge is the time it would take to train a controller with a large number of rules and large size of fuzzy membership graphs. For a design with 3 inputs, 5 fuzzy sets per input and 256-width universe of discourse, the solution space of fuzzy parameters contain approximately 10^{123} solutions. Therefore the solution space is too large for a single algorithm to search in a reasonable amount of time.

Our genetic process attempts to resolve both issues: lack of sufficient training data and wide solution space. We use multiple algorithms to help narrow the scope of the search and limit the tremendous amount of sample data needed to accurately optimize the controller. The next section will explain the flow of our training and optimization process.

4.3.1 - Software Design Overview

Our goal is to use successive genetic algorithms that rely on both a set of narrow training data and a set of broad training data. The key to addressing both challenges mentioned in the previous section is to separate a single iterative algorithm into two separate stages of iterative algorithms. The first stage narrowly explores the solution space for fuzzy parameters by focusing on the contribution of each individual input to the diagnosis of a disease. This is an extension of the objective in [24]. The second stage more broadly explores the solution space for fuzzy parameters guided by the solutions learned in the first stage. The second stage will focus on the combination of all inputs to the diagnosis of the same disease.

Our fuzzy logic controller has three inputs and one output. During the first stage, an iterative algorithm is executed for each input. The solution space of fuzzy parameters for each input contains approximately 10^{15} solutions and 256 output data points. These smaller quantities make the algorithm much more efficient requiring significantly less time and medical training data. The optimized fuzzy membership graphs and rule matrix for each contributing input provide a good foundation to begin the second stage.

During the second stage, a single iterative algorithm is executed for all 3 inputs. The solution space of fuzzy parameters for all inputs contains approximately 10^{123} solutions and 10^{8} output data points. Although the solution space is bigger, the fuzzy parameters learned from the first stage provide some guidance as to where to begin searching. The final fuzzy membership graphs and rule matrix solution is the ultimate genetic solution sent to the hardware fuzzy context register. The diagram below displays our overall process of tuning a fuzzy-based risk evaluator.

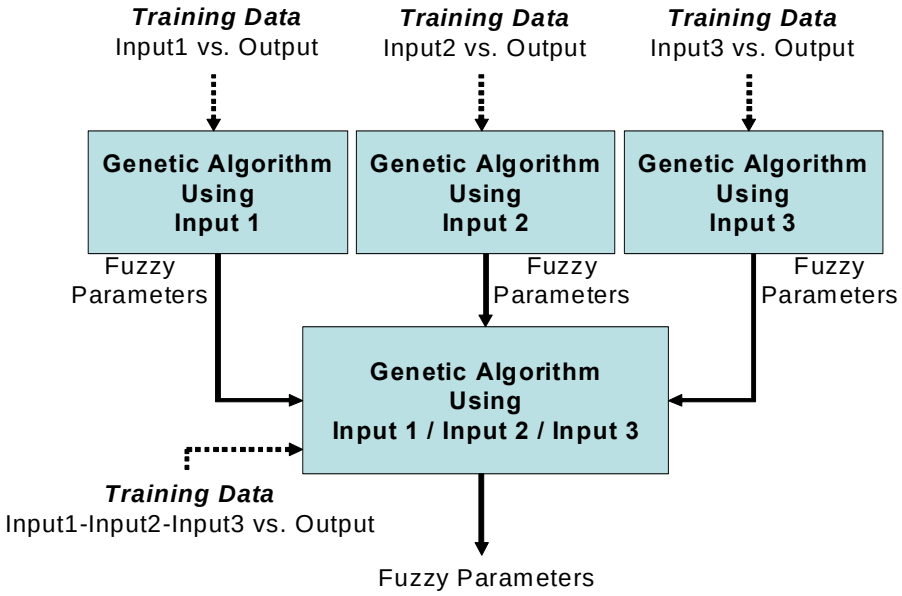

Figure 4.16 – Overall Genetic Training Process

Each genetic algorithm box represents an iterative algorithm dedicated to training fuzzy parameters for specific types of training data. All first stage algorithms require training data that relates an individual input to the output diagnosis. The second stage algorithm requires training data that

relates all inputs to the output diagnosis. The next section will explain design specifications that are defined in the software algorithm.

4.3.2 - Design Considerations

This section will briefly describe some factors that are considered in the design process such as software language, programming platform, structural hierarchy and modularity. The two types of iterative algorithms are designed: one for single input training and the other for multi-input training. Both programs are written in MATLAB and executed on a personal computer. MATLAB provides high level mathematical functions for arrays and complex data structures. The m-file programs are designed to be modular and reusable enabling the programs to be easily called, modified and amended by the engineers. The fitness, selection, crossover and mutation modules must be linked to the top level program and referenced using function calls. During the training process, constant adjustments to these program modules are necessary to get the most accurate and effective specifications. The next section will explain other very important aspects of the evolutionary algorithm like chromosome encoding, fitness function and genetic operators.

4.3.2.1 - Chromosome Encoding Encoding potential solutions into chromosomes is a very important part of genetic algorithms. This section will describe how the fuzzy membership graphs and rule proposition matrix are encoded. The rules are relatively easy to encode. The antecedents and consequents of a rule matrix are represented by one of five fuzzy sets in both

input and output fuzzy membership graphs. These fuzzy sets are labeled with numeric values which are used to encoding and decoding the rules chromosome. The length of the chromosome is determined by the number of rule propositions in the fuzzy controller. In our multi-input iterative algorithm, 3 inputs and 5 fuzzy sets create a rule matrix that contains 125 rule propositions. The diagram shows an example of a matrix with 125 rule propositions. The numeric labels are used instead of linguistic terms.

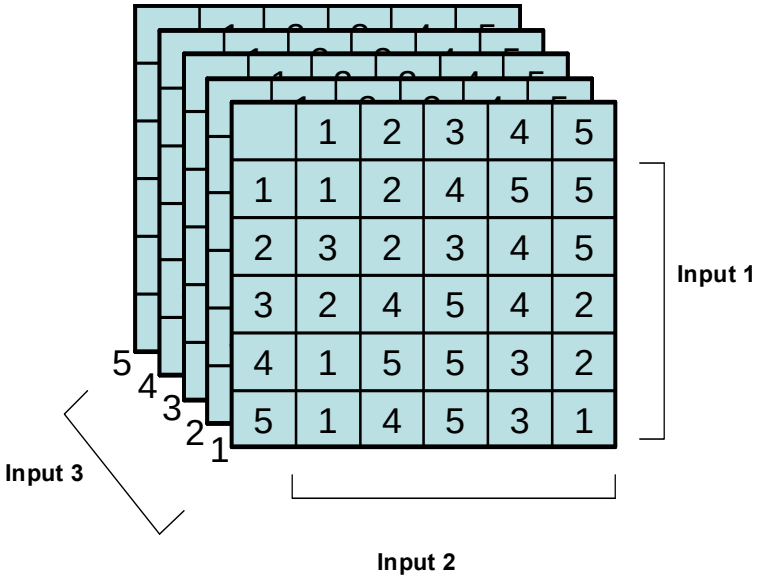

Figure 4.17 – Rule Matrix w/ 3 inputs and 125 rule propositions

When encoding the matrix, we can save memory and complexity by limiting the chromosome content to rule consequents. Since the arrangement of the antecedents never change, the chromosome only requires information about the consequents [42]. The figure below illustrates a portion of the rules chromosome using the first 25 rules of the above matrix.

Figure 4.18 – Partial Chromosome of Rule Matrix

Rule 1: If Input 1 = *1* & Input 2 = *1* AND Input 3 = *1* Then Output = 1

Rule 2: If Input 1 = *1* & Input 2 = *2* AND Input 3 = *1* Then Output =2

:

Rule 25: If Input 1 = *5* & Input 2 = *5* AND Input 3 = *1* Then Output = 1

The location of the rule consequents in the encoded chromosome is determined based on the numeric labels in rule antecedents.

In our single input iterative algorithm, 1 input and 5 fuzzy sets create a rule matrix that contains 5 rule propositions. The diagram shows an example of a matrix with 5 rule propositions. The numeric labels on the left are the input fuzzy sets.

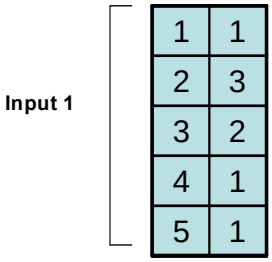

Figure 4.19 – Rule Matrix w/ 1 input and 5 rule propositions

The encoded chromosome is also shown below.

Figure 4.20 – Encoded Chromosome of Rule Matrix

Encoding the fuzzy membership graphs is another important aspect of constructing the iterative genetic algorithm. In our fuzzy design, membership graphs are based on triangular shapes. The triangular fuzzy sets are based on linear equations and functions. The challenge in encoding these graphs is using the chromosome of the least length. In our design, we use improvised forms of fuzzy membership graphs to help simplify their encoding. As displayed in the following diagram, every other fuzzy set starts and end at the common points.

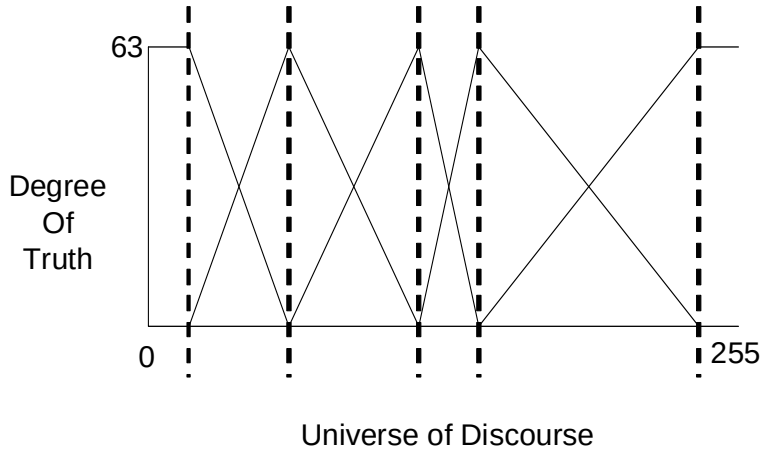

Figure 4.21 – Improvised Fuzzy Membership Graph w/ 5 fuzzy sets

The graph shown above pinpoints these common points of intersection. The chromosome for this membership graph can be encoded using these 5 points of intersection as shown in the following diagram [43].

14	53	121	135	245

Figure 4.22 – Encoded Fuzzy Membership Chromosome

For our multi-input iterative algorithm, three fuzzy membership graphs would require a chromosome of with a length of 15 genes. For our single input iterative algorithm, one fuzzy membership graph requires a chromosome with a length of 5 genes.

The length of each encoded chromosome illustrates the importance of improving our approach to designing the optimization process. For single input iterative algorithms, rule chromosomes and fuzzy membership chromosomes only require a length of 5 genes. For multi-input iterative algorithms, rule chromosomes and fuzzy membership chromosomes require a length of 125 genes and 15 genes, respectively. This decrease in length can improve the effectiveness of the genetic search by decreasing the execution time and decreasing the need for tremendous amounts of training data.

4.3.2.2 - Fitness Function The fitness function of both iterative algorithms uses a variation of the mean square error method to calculate the error between predicted training data and measured fuzzy logic output data. The root mean square error (RMSE) formula retains the same dimension of the error difference between predicted and measured values. The equation below illustrates the how error is derived for each solution.

$$\text{Error} = \sqrt{\frac{1}{N} * \sum_{n=1}^{N} (P_n - M_n)^2}$$

Equation 4.1 – Root Mean Square Error

The P_n variable is the output predicted in a set of N training data. The M_n variable is the measured output from the fuzzy logic controller. To obtain the M_n variable, we enter input data from the set of N training data into a software version of our fuzzy controller. The software version of the fuzzy controller will be outlined later on in this chapter.

Solutions with smaller errors have higher fitness and greater suitability than other potential solutions. To help illustrate this relationship, we use an additional equation to inversely convert error values to fitness levels. Based on the following equation, the smallest error produces a fitness rating of 1000.

$$\text{Fitness} = 1000 * \frac{1}{1 + \text{Error}}$$

Equation 4.2 – Error Fitness Function

The following figures illustrate functional plots of the fitness function. These plots show how a small change in error can exponentially affect the fitness rating. A fitness rating of 100 has a corresponding root-mean-square error of 9, or approximately 3% of the total universe of discourse.

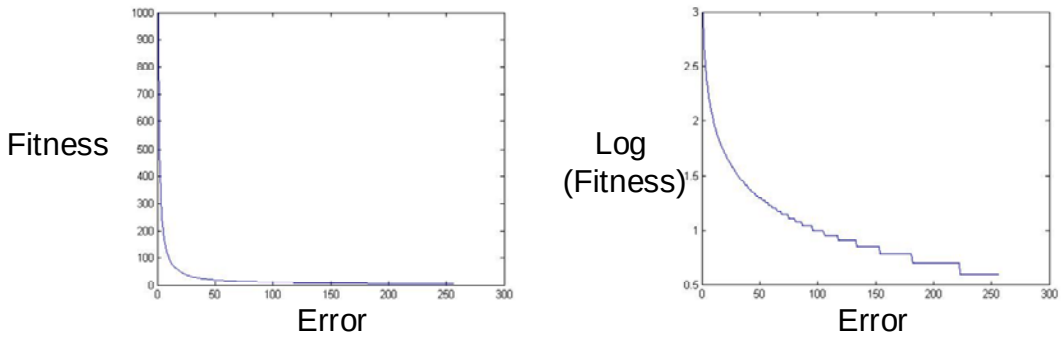

Figure 4.23 – Fitness Function Plots

4.3.2.3 - Genetic Operators Genetic operators perform several important tasks in the process of natural selection. This section will discuss the techniques we utilized in the design of our evolutionary algorithms. The selection, crossover and mutation methods differ between the single and multi-input iterative algorithms. Larger solution spaces often require increased impact from these processes. The choice of specific techniques and their genetic probabilities must be carefully considered and adjusted until the right balance is reached.

Tournament selection is chosen in both algorithms. It requires the least amount of computation and provides greater control to designers over selection process. As stated earlier, this algorithm randomly chooses a group of individuals to participate in a competition with each other. We sequentially select the best candidates based on the pressure probability specified. The selection pressure of the tournament method determines the probability of selecting the best candidate. The higher the pressure, the better the chances are of selecting top performing candidates. After the two

best candidates are selected, they are transferred to the crossover phase. Elitism is also employed with the beginning of each new genetic process.

For the single-input iterative algorithms, one-point crossover and one-point mutation are chosen for reproduction. The multi-point crossover and multi-point mutation are chosen for multi-input iterative algorithm. The crossover probability determines how often reproduction occurs between selected individuals. This probability is usually around 80%. The mutation probability determines how often random genetic alteration occurs in individuals. This probability is usually under 1%.

Another variable that must be tracked in the genetic process is maturity. It is very important that the maturity of the population is measured to ensure diversity and avoid premature convergence. In our algorithm, maturity is measured in two ways either by determining how many common genes exist in a population or by calculating the ratio between the mean and maximum fitness in a population. By keeping track of the trends in genetic evolution, we can identify when the population begins to converge. In our iterative algorithms, the stopping condition for fuzzy membership training or fuzzy rule training is met when a convergence point is reached. This convergence point represents a percentage of the population which has a common set of genes. These percentages can differ depending on probabilities of other genetic operators.

For example, if crossover and tournament selection probabilities are high and mutation probabilities are low, then premature convergence is expected to occur. High tournament selection and high crossover probabilities forces population to quickly converge to the one set of chromosomes and low mutation probabilities reduces the possibility for those chromosomes to change. This scenario would affect the way we

calibrate our convergence point. Each genetic probability and convergence point has a direct and indirect affect on the evolutionary process. It is important to establish the right balance between each operator variable.

4.3.3 - Program Architecture

Our software architecture is designed in a modular hierarchy. The block diagram below provides a high level description of the architecture.

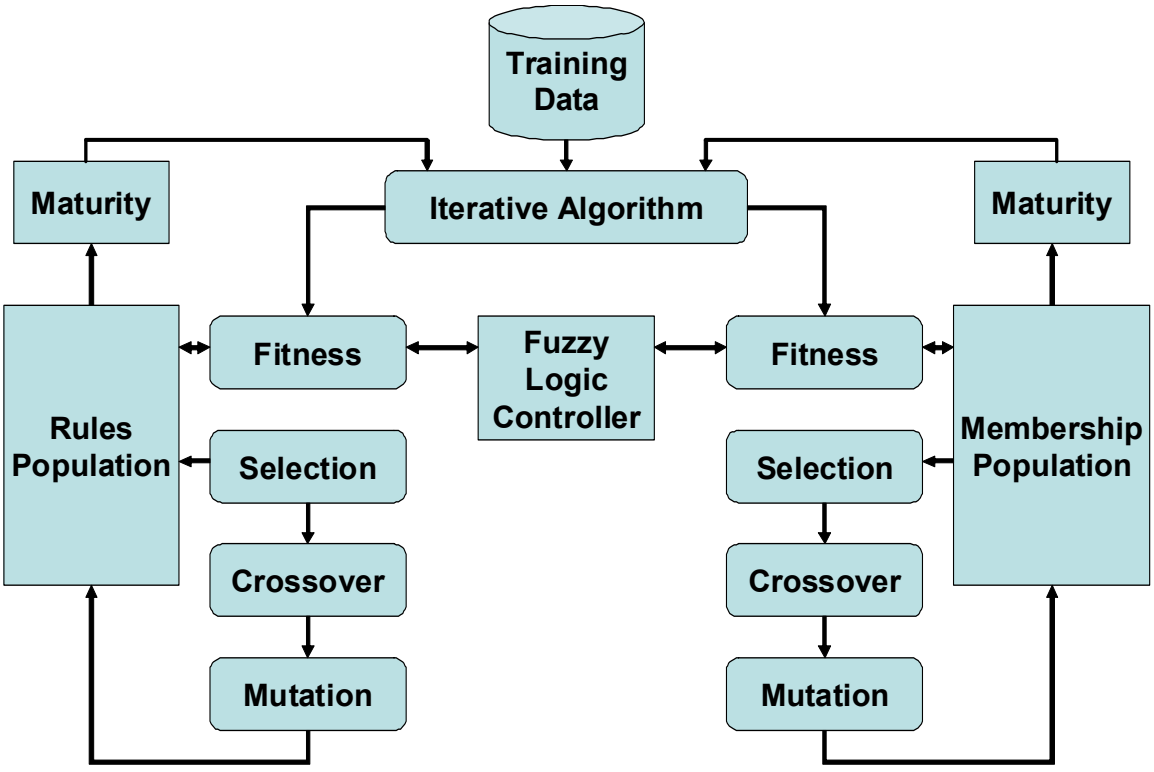

Figure 4.24 – Software Block Diagram

The iterative algorithm module is the top level program of the architecture. The above structure illustrates how modules are interrelated with each other.

Both single-input and multi-input algorithms possess this architecture with minor differences in the content of the training data and size of the encoded chromosomes. The next sections will discuss algorithm module, data type and internal function contained in the architecture.

4.3.3.1 - Iterative Algorithm The iterative algorithm module initializes the algorithm conditions, requirements and specifications such as population size, chromosome lengths, fuzzy logic system constraints, genetic operator probabilities, maturity convergence points and the initial random membership solution. The initial random solution allows the rules layer search algorithm to run on the first iteration. The iterative module has several responsibilities which include:

- Storing membership and rule populations and training datasets
- Evaluating population fitness and maturity ratings
- Executing genetic algorithm using function calls to genetic operators for fuzzy membership and rule search layers

The execution flow of the iterative algorithm module is shown below. Figure 4.23 illustrates how initial rules layer algorithm requires an encoded membership solution to complete the fuzzy logic fitness function. In a single-input algorithm, this solution is completely random and as a result the starting point is completely random. However, in a multi-input algorithm, starting at a random point makes the search very difficult and time consuming since the solution space and possible data point are much larger. The encoded fuzzy membership solution for each single-input algorithm will

serve as the starting point for the multi-input search algorithm. This change is the major difference the single and multi-input training algorithms.

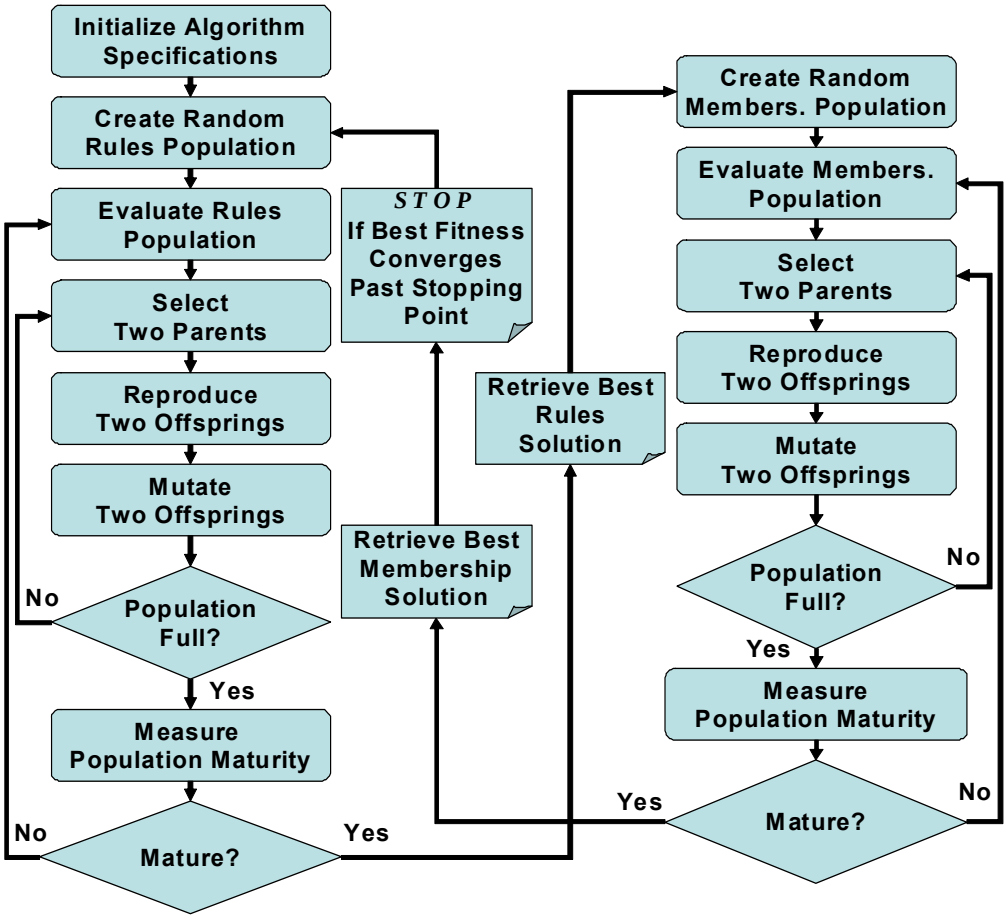

Figure 4.25 – Iterative Algorithm Flowchart

The algorithm terminates when the final maximum fitness value from the optimizing the fuzzy membership solutions begins to converge. This prevents the algorithm from running forever. In order to detect this convergence, the fitness rating of the best membership solution is stored and compared to the subsequent fitness rating of the best membership solution. The error between the two successive fitness ratings is compared to a final

stopping point value. This value represents the maximum allowable error between the best solutions. If this error falls below the final stopping point, the algorithm terminates. To prevent endless execution and avoid premature termination, this stopping point is tested and analyzed in the next chapter.

4.3.3.2 - Fitness Module To determine fitness ratings, the fitness module calls the fuzzy logic controller as a function to retrieve measured output data based on inputs stored in the training dataset. The top level iterative module transfers the fuzzy parameters through the fitness module. They are added as arguments in fitness function calls and include:

- Fuzzy membership and rules parameters
- Input and output variable dimensions
- Training dataset sample input-output pairs

The module returns a fitness rating for a solution candidate to be stored in its respective memory. The following diagram illustrates the module's execution flowchart.

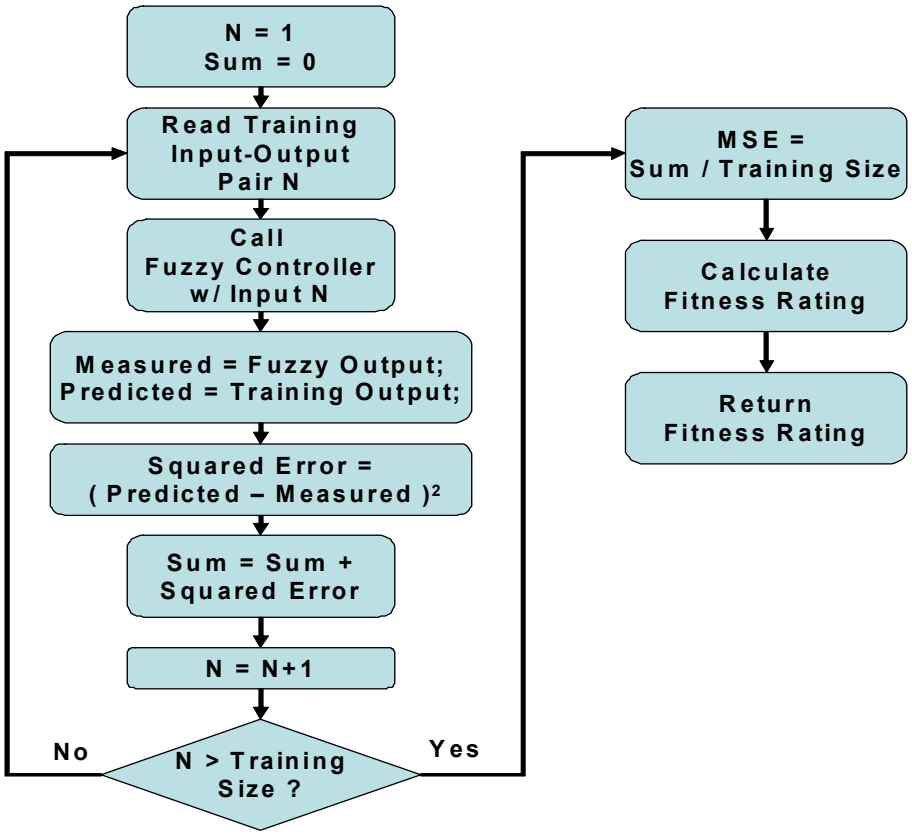

Figure 4.26 – Fitness Module Flowchart

4.3.3.3 - Selection Module The selection module chooses individuals from the solution population for reproduction. The module is called as a function in the top level iterative algorithm and accepts arguments such as the population memory and their fitness ratings. The tournament technique is used to select appropriate solution candidates. The following flowchart demonstrates the execution flow of the module.

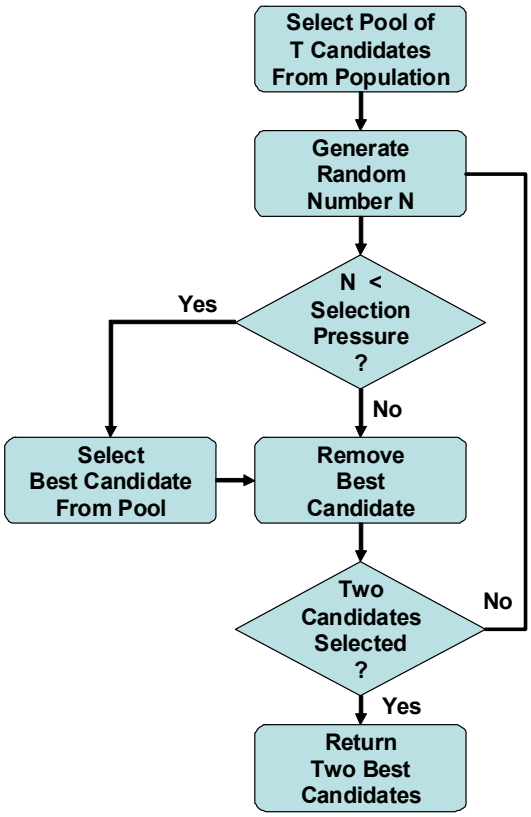

Figure 4.27 – Selection Module Flowchart

The tournament size and selection pressure are genetic specifications initialized in the iterative algorithm module.

4.3.3.4 - Crossover Module To produce new solutions, the crossover module cuts and blends certain genes of parent chromosomes. The top level iterative algorithm calls the crossover module as a function by passing selected individuals. One-point and two-input crossover methods are employed for single-input and multi-input iterative algorithms, respectively. The figure below is a simple flowchart of the software.

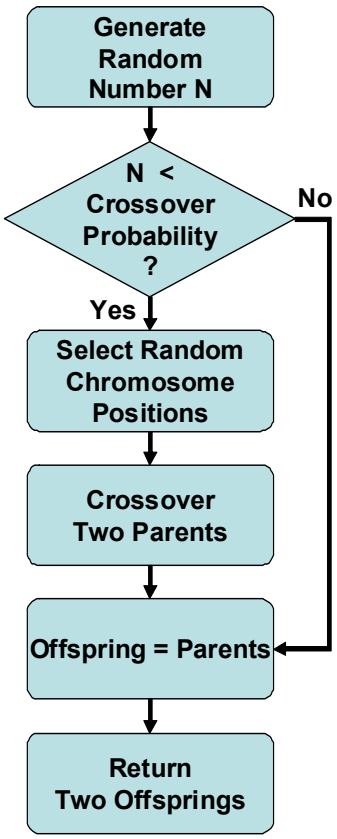

Figure 4.28 – Crossover Module Flowchart

The genetic probability for crossover is set by the top level iterative algorithm. The number of crossover positions selected depends on the type of method being used for the algorithm.

4.3.3.5 - Mutation Module The mutation module is responsible for resetting certain genes within chromosomes. The function is called by the iterative algorithm with the new offspring solutions as arguments. The location of the mutated gene is chosen randomly. The following diagram illustrates the mutation process.

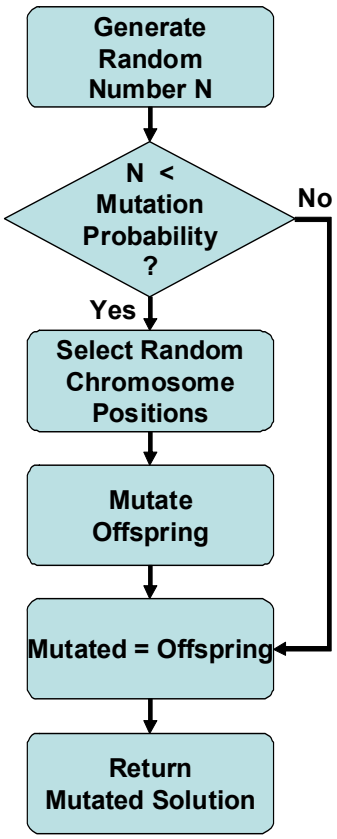

Figure 4.29 – Mutation Module Flowchart

The range of the mutation is based on the number of possible gene values. The number of mutation positions depends on the type of iterative algorithm being used. The mutation probability is set by the top level algorithm specifications.

4.3.3.6 - Maturity Variable The maturity variable measures the convergence point by comparing the candidate solution with the highest maximum fitness to other candidates in the population. A percentage of genes in each chromosome are compared to measure how many candidates possess similar traits. Once a threshold of similarity is reached, the

algorithm switches to optimizing the next fuzzy parameter. There are many ways to measure the maturity of the population. In our design, we focus on two key population properties. One method is to determine the number of individuals that have a percentage of common genes. MATLAB offers great internal functions that enable us to compare and count the number of equal genes between the chromosomes in the population and the chromosome with maximum fitness. This code appears in the top level iterative algorithm for both fuzzy member and rule optimization. The flowchart for calculating maturity is shown below.

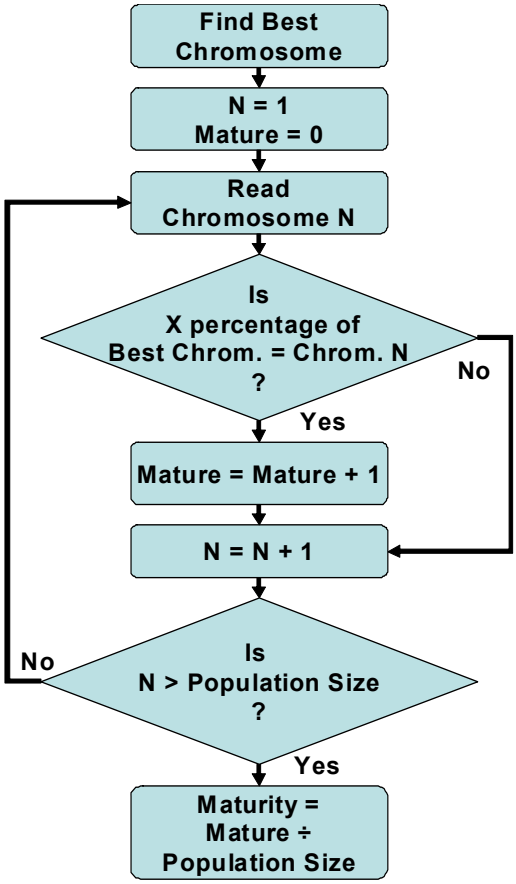

Figure 4.30 – Maturity Calculation Flowchart

Another popular method simply measures the ratio between mean and maximum fitness of a population. Although the genetic similarities between individuals in a population are not known, the ratio provides an added knowledge of the diversity between their capabilities (or fitness). This code also appears in the top layer.

Maturity Ratio = Mean (Generation X) / Maximum (Generation X)

Equation 4.3 – Maturity Ratio Formula

4.3.3.7 - Fuzzy Logic Controller The fuzzy logic controller module is a software version of the hardware architecture presented in the hardware design section. The fuzzy logic controller is called as function by the fitness module using inputs taken from training data. The function accepts sample inputs and returns the measured output value for evaluation. Within the software code, function calls are made to fuzzify, rule, minimum, mux and defuzzify modules. The fuzzy function also accepts the encoded fuzzy parameters from the fitness function module and allowing the controller to configure the fuzzy operation to the specifications contained in the encoded parameters.

The fuzzify function uses encoded membership chromosomes to calculate the proper fuzzy labels and degrees of truth for the input training samples. The rule and minimum functions use this information to determine the rule consequents and firing strengths of the rule propositions. These rule propositions are also provided via the encoded rule chromosomes transferred from the fitness module. The mux and defuzzify functions combine this

information to form an aggregate graph and its corresponding center of area output value. This value is then returned to the fitness function. The flowchart for the fuzzy logic controller is displayed below in a horizontal format.

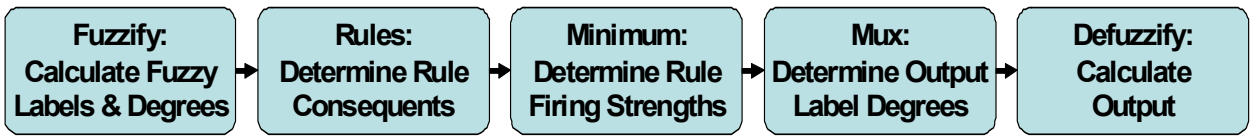

Figure 4.31 – Fuzzy Logic Controller Flowchart

Each file is written as separate programs and is called sequentially to implement the fuzzy operation. This program also provides an excellent method of comparing output data to the hardware based fuzzy logic engine implemented in our research. The program is significantly slower than the parallelized hardware version, which causes our training algorithm to consume a great deal of time.

4.3.3.8 – Stopping Conditions Evolutionary algorithms require many nested loops to simulate repeated natural patterns and processes. Stopping conditions serve dual purposes in measuring the progress of the genetic search and ensuring that these loops do not unexpectedly run forever. When selecting appropriate stopping conditions we have to be sure that we prevent any premature stoppage and avoid time consuming endless loops. Population maturity levels, referenced by the mature variable, help to observe the progress of each parameter being optimized, which can either be the rules and membership functions. However, in some cases the maturity

variable stalls which causes a never-ending loop to occur for the corresponding parameter. To prevent this, an additional condition was added which terminates the loop if the loop consumes a large percentage (approx. 10% – 20%) of the maximum generations designated for the entire algorithm.

The final stopping condition for the overall iterative algorithm is met when one of two requirements are satisfied. One condition states that if the overall algorithm surpasses a maximum number of generations, the program will terminate. The other condition states that if after a certain number of attempts you cannot locate a solution with a greater fitness rating than the maximum fitness by a certain margin, the program will terminate. These stopping conditions allow the algorithm to search for the best solutions without terminating prematurely or looping endlessly. They were also important due to the large amount of execution time required to perform the slower fuzzy logic software program many times over many generations.

4.3.4 - Data Structures

Array data structures are used to implement many different types of memory in the program architecture. Genetic chromosomes, candidate solution populations and training datasets are based on this structure. A genetic chromosome is represented by an array of integer values, each symbolizing a genetic trait of a candidate solution. The length of the array is equal to the length of the chromosome. The fuzzy membership chromosome contains up to 5 integer values which range from 0 to 255 (universe of

discourse). The fuzzy rule chromosome contains integer values of various lengths which range from 1 to 5 (maximum number of output fuzzy sets).

Solution populations for both fuzzy memberships and rule propositions store a pool of genetic chromosomes. Two-dimensional array objects are instantiated in the top level iterative algorithm to represent these populations. The size of the array is based on the size of the population and the length of the chromosome. These populations are initialized at random at the beginning of the algorithm. These population arrays are modified as new generations of candidates are generated.

Training data is also entered and initialized in an array structure. This dataset stores a wide variety of sample inputs along with their predicted output response. The length of the array is equal to the total number of inputs and outputs specified in the dataset. The width is the number of data samples provided. The data is referenced by the fitness module to calculate error and fitness ratings of solution candidates.

4.3.5 - MATLAB Implementation

The software is written in MATLAB and executed on a desktop computer. This section will review the MATLAB files designed to implement the software architecture described in the previous section. The top level file for the iterative algorithm is ***iterative.m***. This file contains the variables for both algorithm and fuzzy controller specifications required to switch between single-input to multi-input iterative training. These specifications are listed below.

Algorithm Specifications:

- Membership & Rule Chromosome Length
- Membership & Rule Population Sizes
- Membership & Rule Selection Pressures
- Membership & Rule Crossover Probabilities
- Membership & Rule Mutation Probabilities
- Membership & Rule Maturity Convergence Points
- Final Stopping Point (Maximum Error)
- Sample Training Data

Fuzzy Specifications:

- Number of Inputs
- Number of Input Dimensions or Fuzzy Sets
- Number of Outputs
- Number of Output Dimensions or Fuzzy Sets

The program files called by our top level software file, ***iterative.m***, are listed below along with a brief description of their responsibilities.

- ***rule_fitness.m***
    - Determine the fitness of rule solution chromosomes
- ***rule_selection.m***
    - Select two rule solution chromosomes for crossover
- ***rule_crossover.m***
    - Generate two new rule solution chromosome offspring
- ***rule_mutation.m***
    - Mutate two offspring to for next generation population

- ***member_fitness.m***
    - Determine the fitness of membership solution chromosomes
- ***member_selection.m***
    - Select two membership solution chromosomes for crossover
- ***member_crossover.m***
    - Generate two new membership solution chromosome offspring
- ***member_mutation.m***
    - Mutate two offspring to for next generation population
- ***fuzzy.m***
    - Conduct fuzzy operation on sample input returning output
- ***fuzzify.m***
    - Find fuzzy labels and degrees of sample input data
- ***rules.m***
    - Output rule consequents of matrix propositions
- ***min.m***
    - Solve for firing strength of rule consequents
- ***mux.m***
    - Determine output degrees for each output fuzzy set
- ***defuzz.m***
    - Calculate output value from center of area method

These files are compiled and executed in the MATLAB environment. The user directly interfaces with the software and directly modifies the algorithm and fuzzy logic specifications labeled in the first section of the software.

Final fuzzy parameter solutions are located by reading the variables designated for ***optimized_members*** and ***optimized_rules*** specified in MATLAB.

4.4 - HARDWARE-SOFTWARE DESIGN SUMMARY

Our design proposal of the hardware fuzzy logic controller and the software iterative training algorithms enable the development and implementation of an efficient and effective medical risk evaluator. The process of diagnosing patient risk is a highly complicated task often requiring the consideration of a wide range of medical factors. However, by taking these small steps to designing a controller that can successfully measure risk, based on existing medical surveys and sample datasets, the possibility of automating this process becomes significantly greater.

The process of developing a risk evaluator relies heavily on the availability of existing surveys which provide risk assessments and statistical prevalence of various diseases based on specific medical categories. Our improvised training process attempts to reduce the number of medical data required to optimize our fuzzy logic device. To accomplish this, our training data includes samples which denote the contribution of each medical input and its impact on the final diagnosis. Therefore, it is important to retrieve research based evidence that studies the prevalence of the disease based on each individual input. This prevalence research is used to offer predicted sample training data in single-input iterative algorithms to help narrow the

genetic search. In the next chapter we demonstrate how this process works for cardiac arrhythmias using medical electro-cardiographs (ECG) and other patient physiological data like age and body-mass index as input factors.

Our hardware design focuses on limiting design costs and execution time. By limiting the number of rule memory and minimum unit, we save considerable power and area. Even though center of area defuzzification method requires considerable resources, we were able to show examples of efficient implementations of our aggregate and division techniques. The mass and area of the aggregate graph is computed in one function. Splitting this operation to multiple output aggregate units by graphical regions improves execution time. Also, our partial-reciprocal multiplication method enables exponentially faster division calculations. The next chapter will present functional simulation and testing results of the system discussed in this chapter.

5.0 – SIMULATION & TESTING

The purpose of this chapter is to present functional simulations that were performed for both the hardware and software components of the overall design. As outlined earlier, the hardware component contains the fuzzy controller, while the software component contains the evolutionary training algorithm. Hardware simulations will ensure functional accuracy and system timing improvements. Software simulations can help validate and verify the training algorithm specifications and setup we applied to our approach. This chapter will also display the result of performance and testing analysis of both design components.

5.1 - SIMULATION APPROACH AND APPLICATION

Our two design components require separate approaches. To evaluate the hardware design, discrete event-driven simulations are performed using the Modelsim Simulator tool. We can enter sample inputs, trigger the fuzzy

operation, analyze internal hardware modules and record the output response. To test the software design, agent-based simulations evaluate the evolutionary software algorithm where individual entities (chromosomes) possess characteristics and behaviors which influence the process and operation of the algorithm. The Matlab programming environment provides excellent interfaces to debug specific code or analyze individual data structures.

Our controller application primarily focuses on the medical diagnosis and risk evaluation of a common cardiac arrhythmia known as sinus bradycardia. According to the American Heart Association, sinus bradycardia occurs when a patient's heart rate is considered to be too slow. However, accurate diagnosis of bradycardia requires the inclusion of several other factors such as age and physical condition. Our fuzzy logic-based risk evaluator will determine a patient's risk level based on three sets of medical data: heart rate, age and body-mass index.

5.2 – HARDWARE SIMULATIONS

The fuzzy hardware system will be tested using event driven simulations. Context registers containing reconfigurable data will be loaded and synthesized externally from the controller module. The context register will include information such as fuzzy membership parameters, fuzzy set dimensions and rule matrix elements. We will simulate 1-input, 2-input and 3-input fuzzy systems containing rule matrix containing up to 125 rule

propositions. Sample inputs will be entered while internal hardware signals are monitored and analyzed for performance and timing. Each input and output utilizes an 8-bit bus with a resolution of 256 values. The next section will discuss how scaling factors were considered when applying heart rate, age, body-mass index and percentage variables to the fuzzy design. The following diagram illustrates the test-bench simulation setup.

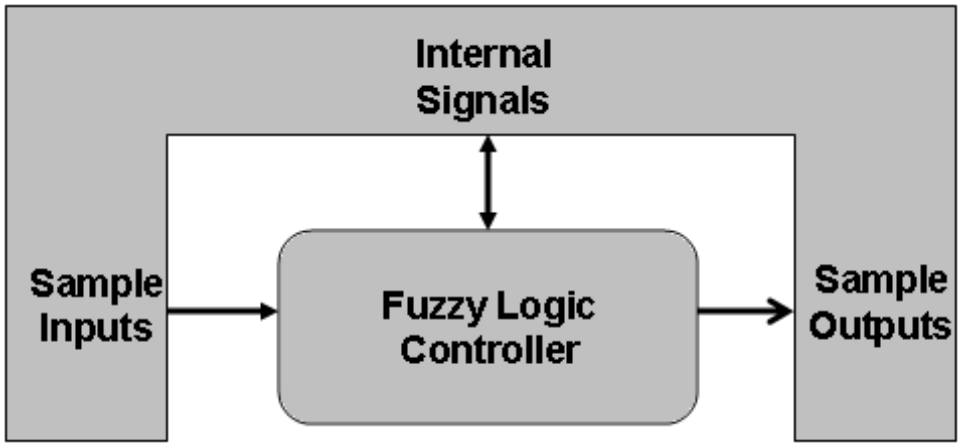

Figure 5.1 – Test-bench Simulation Setup

5.2.1 - Scaling Factor

The quantifiable medical and physiological measurements are processed by the fuzzy system and generate an output response in the form of a percentage rating. The range of each input medical data and output percentage rating is shown below.

- Heart Rate – 0 to 255
- Age – 0 to 127
- Body Mass – 0 to 50

- Percentage Risk – 0 to 100

Each variable must be properly scaled to the 8-bit data-bus in the fuzzy hardware system. The heart rate variable has scaling function of (1x), the age variable has a scaling function of (2x) and the body mass index variable has a scaling function of (5x).

To properly scale the output variable range we must consider the arrangement of the output fuzzy membership graph. The shape of the output graph has a direct influence on what the highest and lowest possible defuzzified value can be. Including this membership graph into the genetic search algorithm would require too much added complexity and time. Instead of searching for an optimal output membership graph, we will use a fixed membership function and scaling factor to simplify the algorithm. The membership graph will have a maximum defuzzified value of 219 and minimum defuzzified value of 19. The scaling function is (2x+19).

5.2.2 – Pre-synthesis Simulation

We will begin by displaying the simulation waveform form a one-input fuzzy controller. The single input variable will contain 5 fuzzy sets and 5 rule propositions which are all driven as input specifications to the controller via their respective input ports. The diagram below shows the fuzzy membership graph and fuzzy rules selected for a one-input simulation. The encoded membership = {31 60 106 108 213} and the encoded rules = {5 3 2 2 1}.

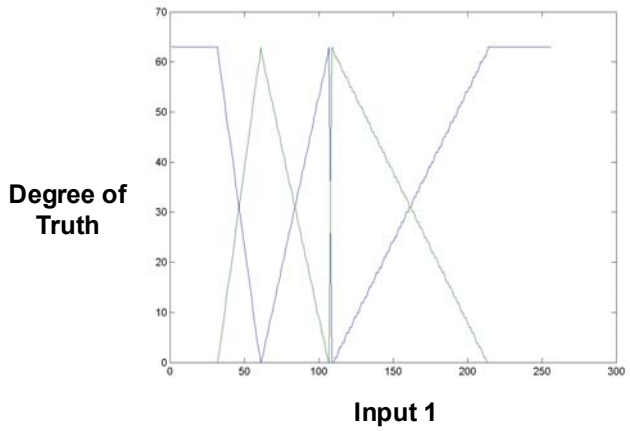

Figure 5.2 – Fuzzy Membership Graph – Input 1

If	*1*	*2*	*3*	*4*	*5*
then	5	3	2	2	1

Figure 5.3 – Fuzzy Rule Matrix

The output membership graph, as mentioned earlier, was specifically designed to match the scaling concerns noted in the previous section. The encoded output membership = {10 60 110 160 210} and the graph is shown below.

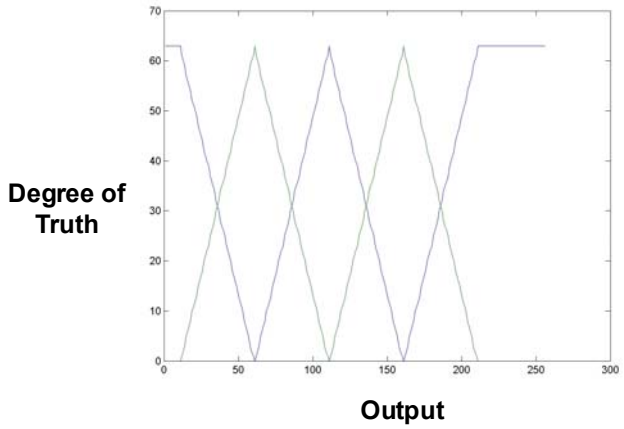

Figure 5.4 – Membership Graph for Output

As soon as the simulation is activated the controller will begin to optimize the fuzzy context memory blocks (input fuzzify, rule matrix and output aggregate blocks). This conversion and memory write operation occurs simultaneously for each module affected. Figures 5.5-5.7 illustrate the simulation waveforms depicting the optimization phase and the relevant internal signals associated with the process. In the current simulation, three fuzzy converters translate context data into one input and output membership graph and rule matrix memory data as depicted in Figure 4.2.

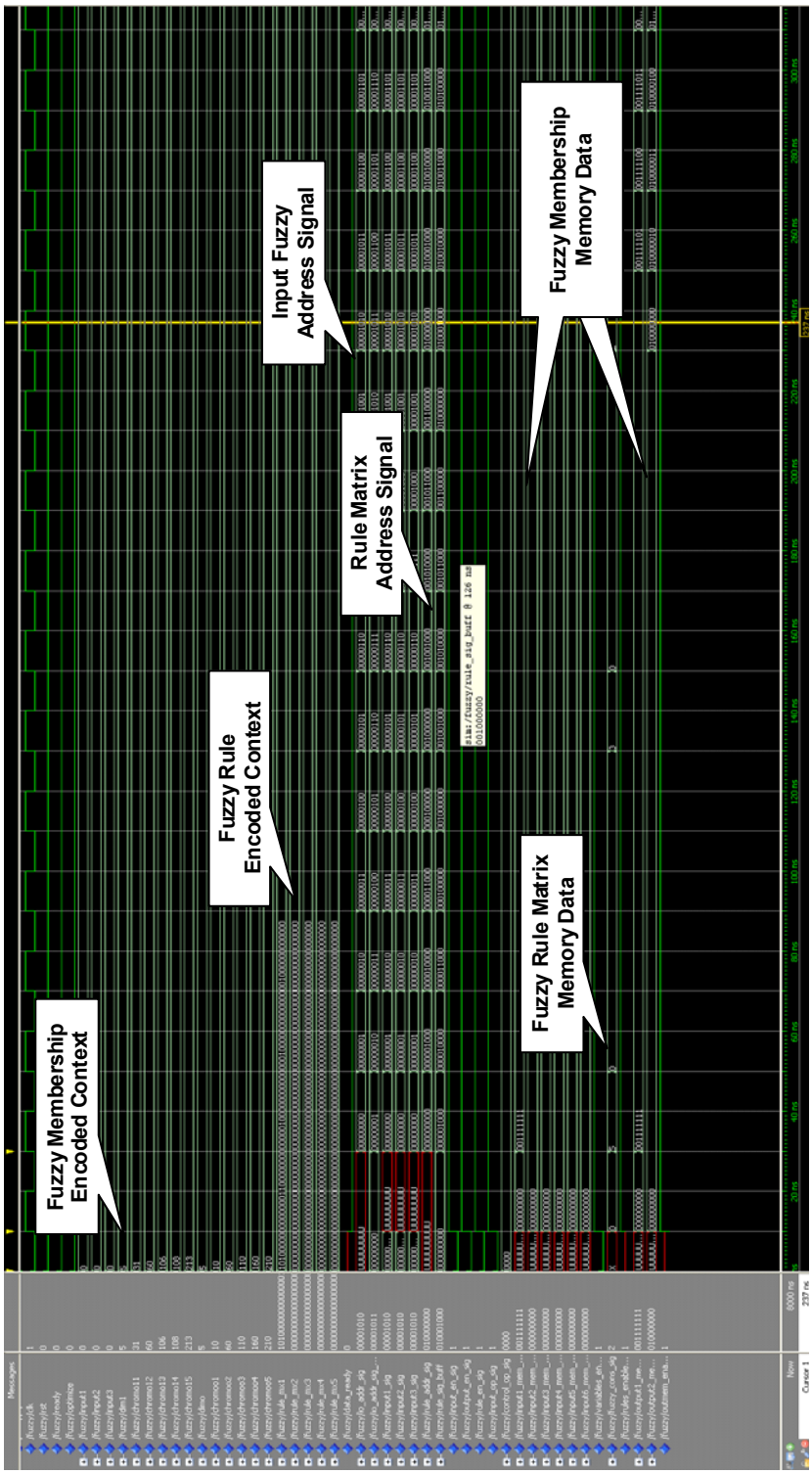

Figure 5.5 – Context Optimization Simulation Waveform 1

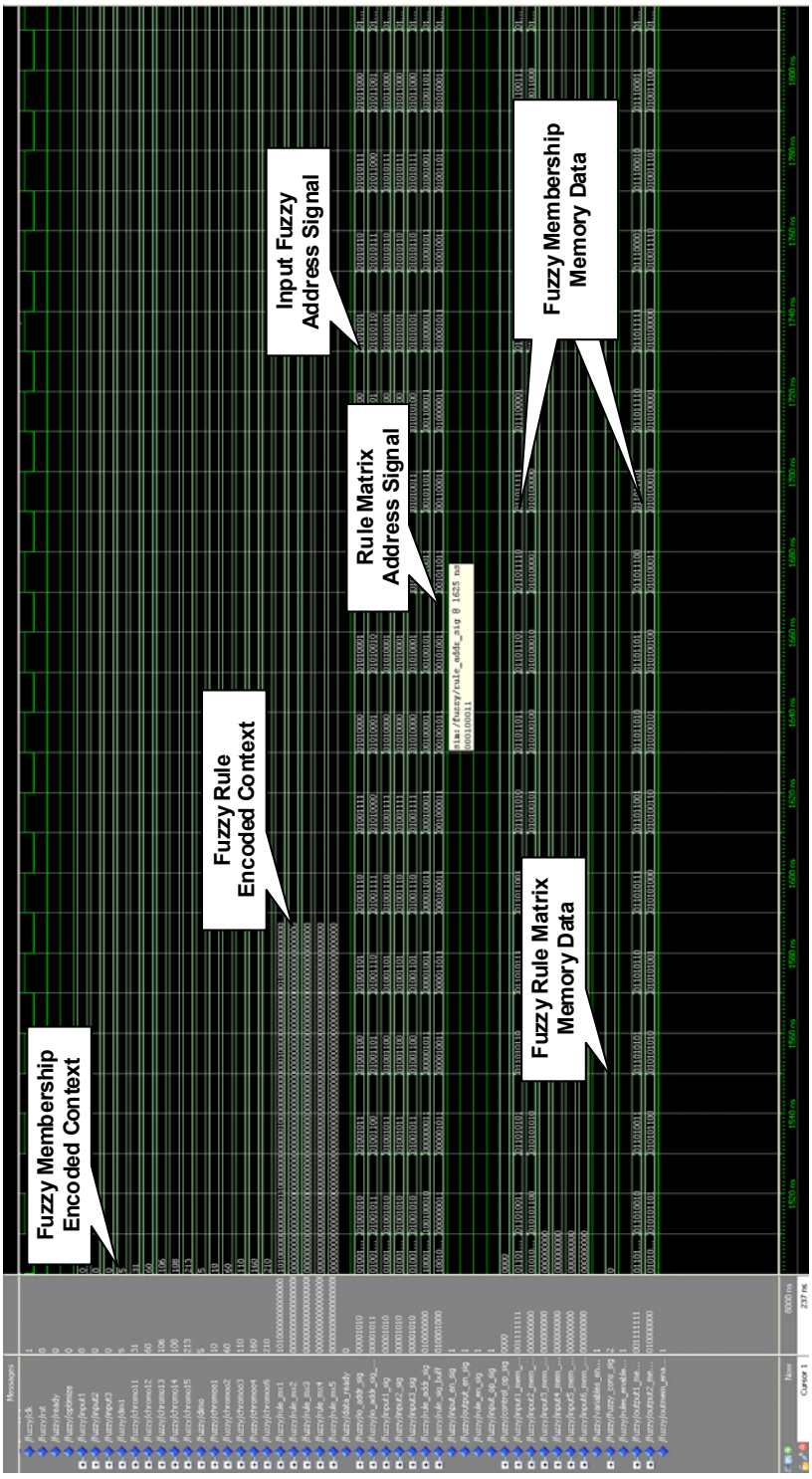

Figure 5.6 – Context Optimization Simulation Waveform 2

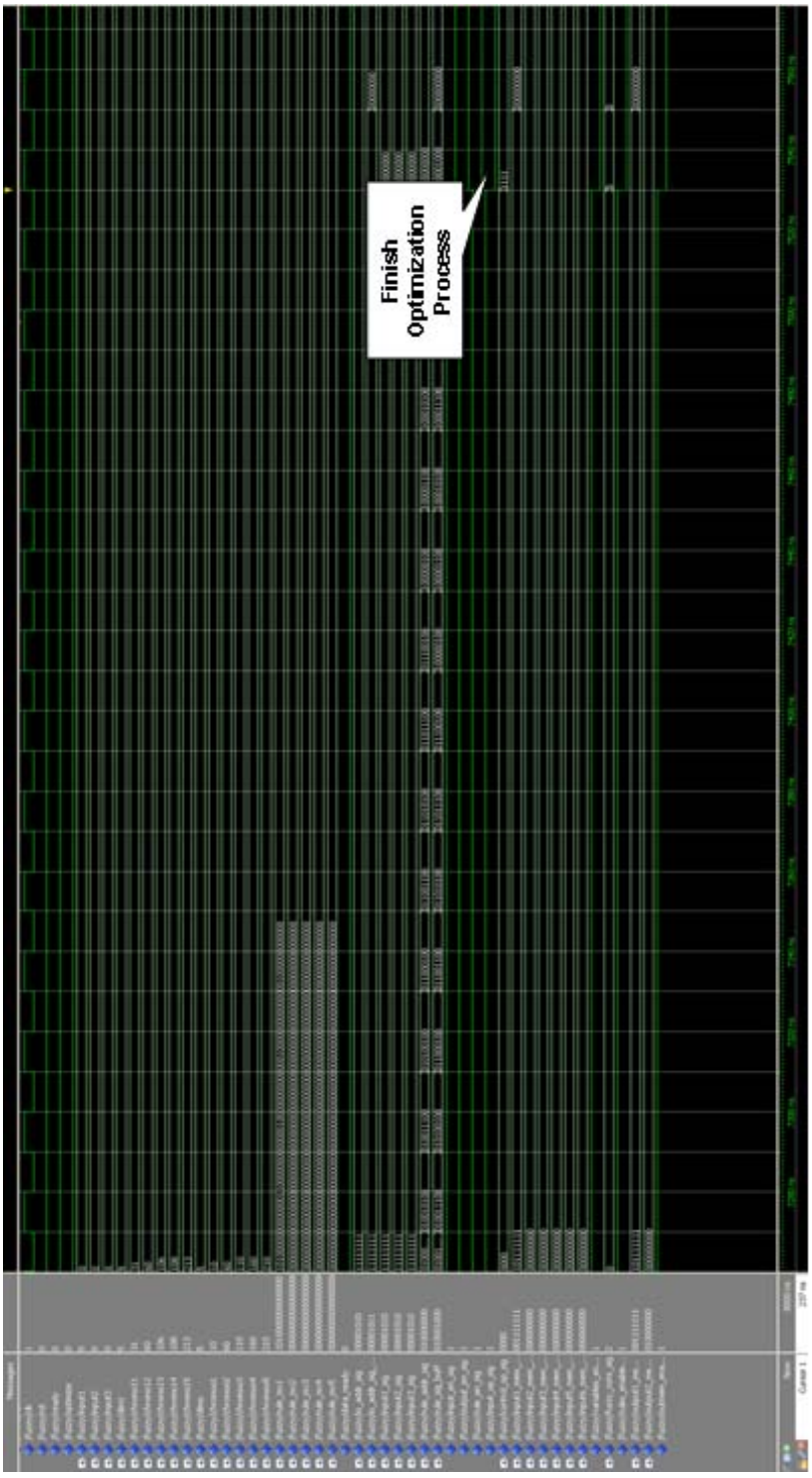

Figure 5.7 – Context Optimization Simulation Waveform 3

In the simulation waveforms, we display three different sections of the optimization process: initialization, translation and completion. The first waveform, Figure 5.5, displays the context data that is being transferred into the converters. The second waveform, Figure 5.6, displays the appropriate input and output membership memory data and rule memory data is transferred to each respective memory modules. The final waveform, Figure 5.7, displays the functional signals of the control unit switching from optimization mode to execution mode. Once the optimization phase is complete, the device switches to execution and waits for input values to be triggered by the user run button. The next step was to input real values and trigger the controller to determine if the data is accurate. We will input three values {20 50 140} and display the simulation results below.

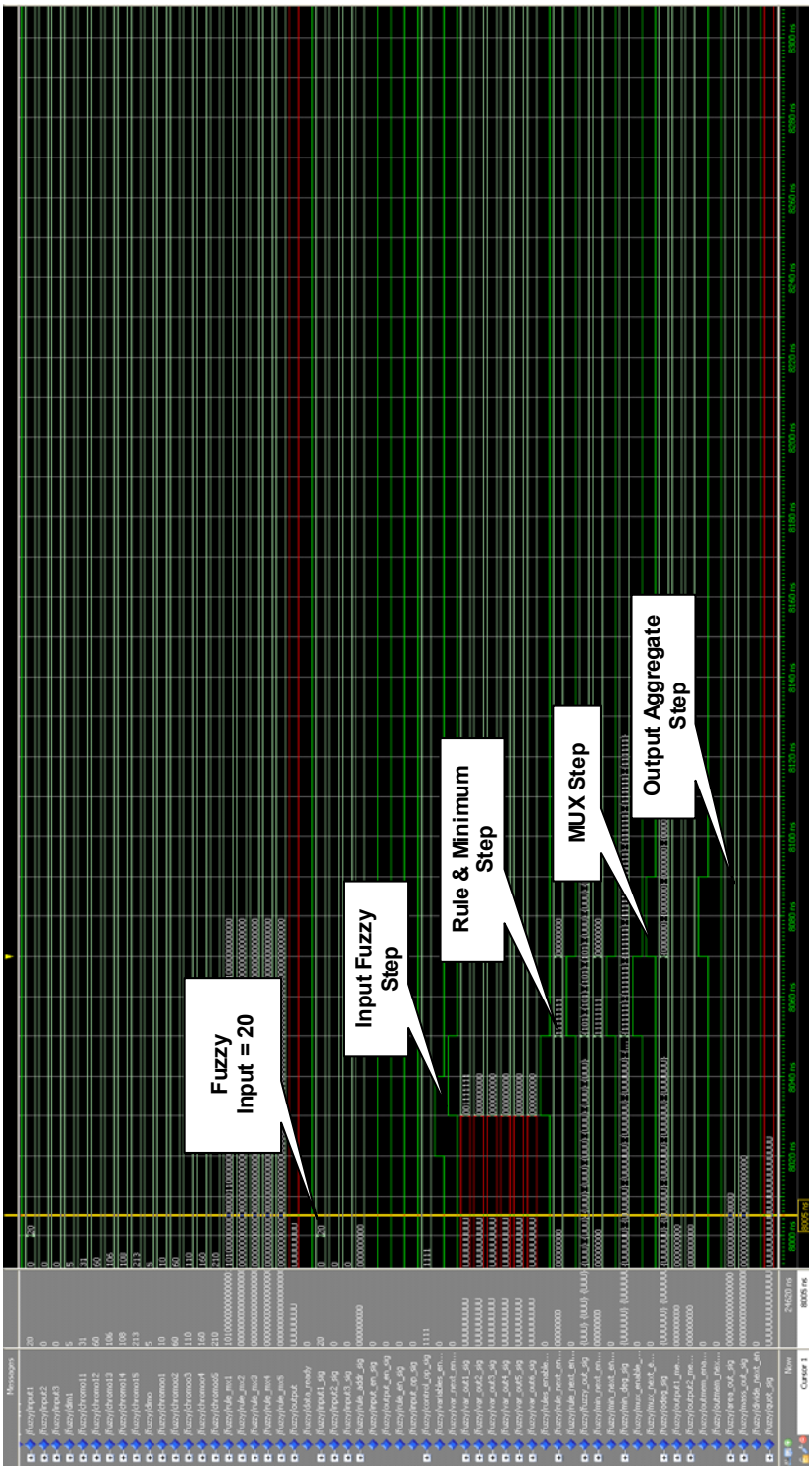

Figure 5.8 – Fuzzy Operation w/ Input = 20

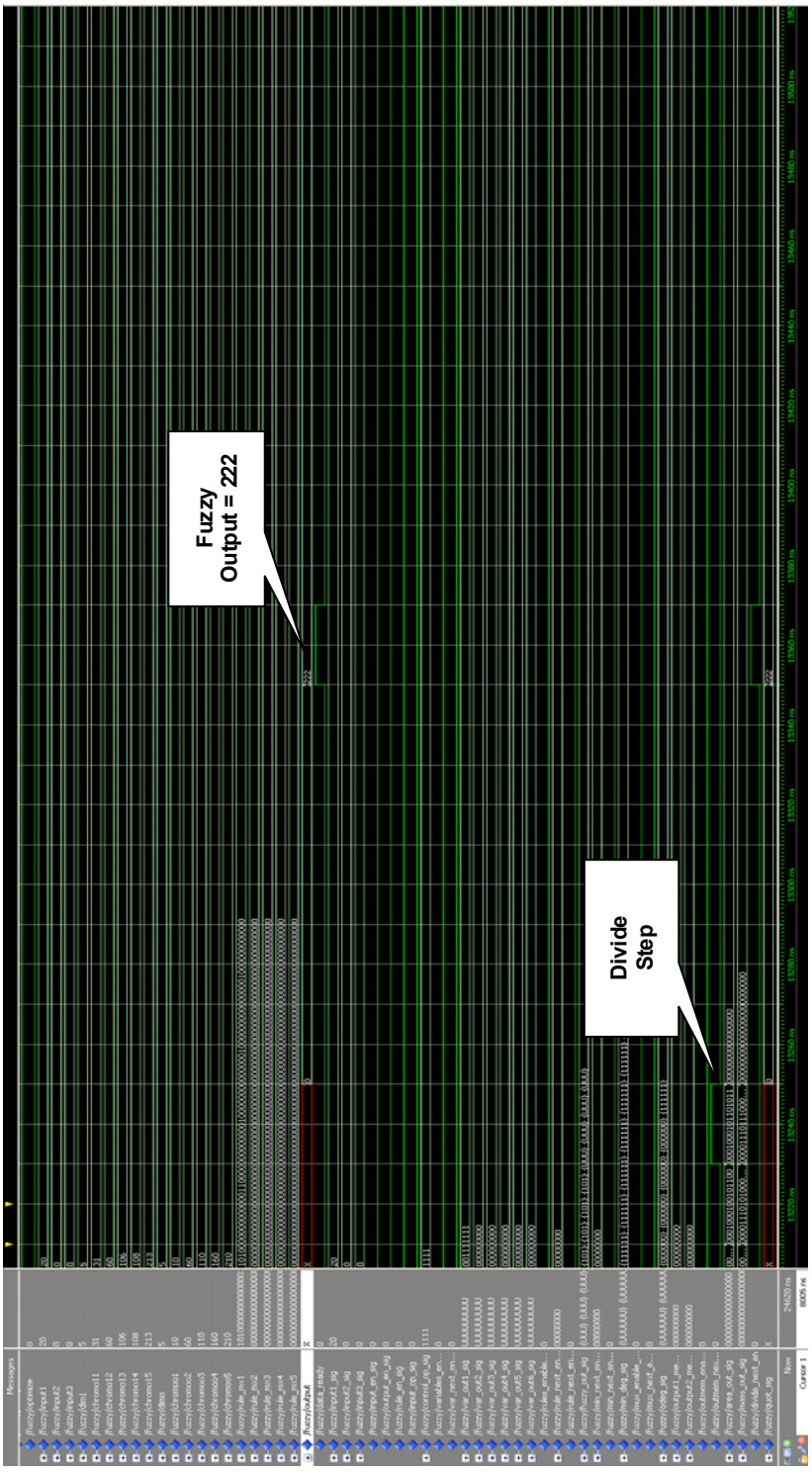

Figure 5.9 – Fuzzy Operation w/ Output = 222

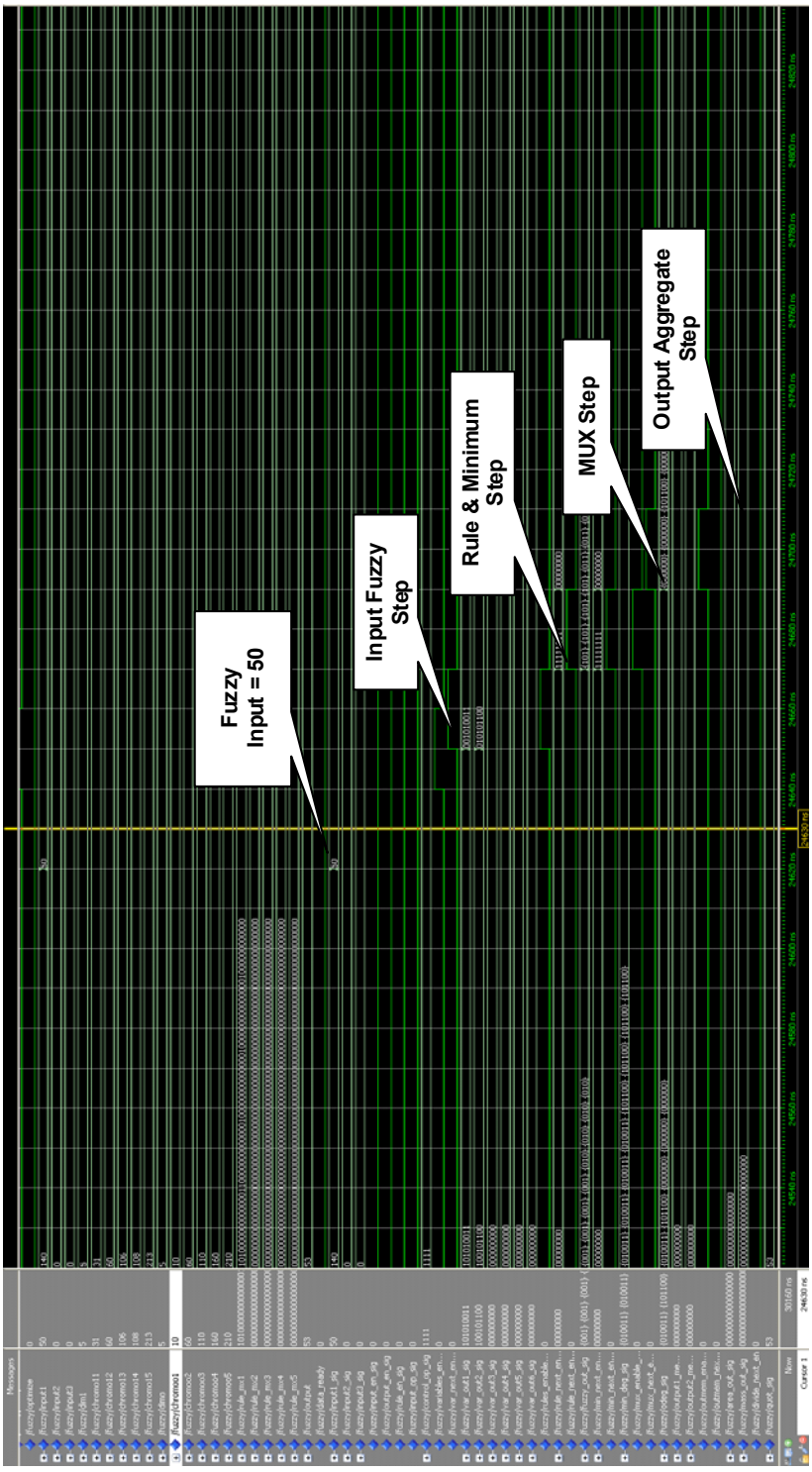

Figure 5.10 – Fuzzy Operation w/ Input = 50

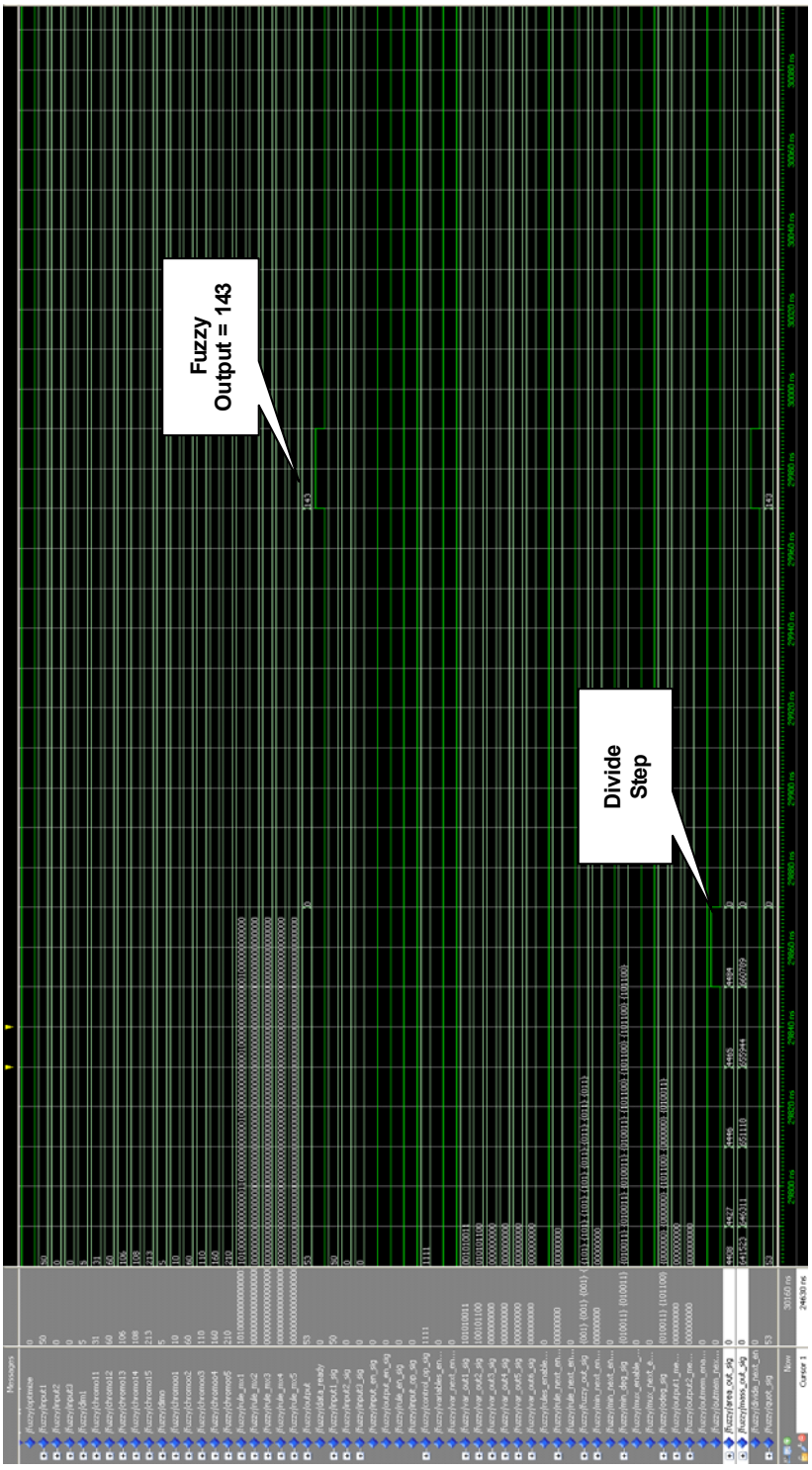

Figure 5.11 – Fuzzy Operation w/ Output = 143

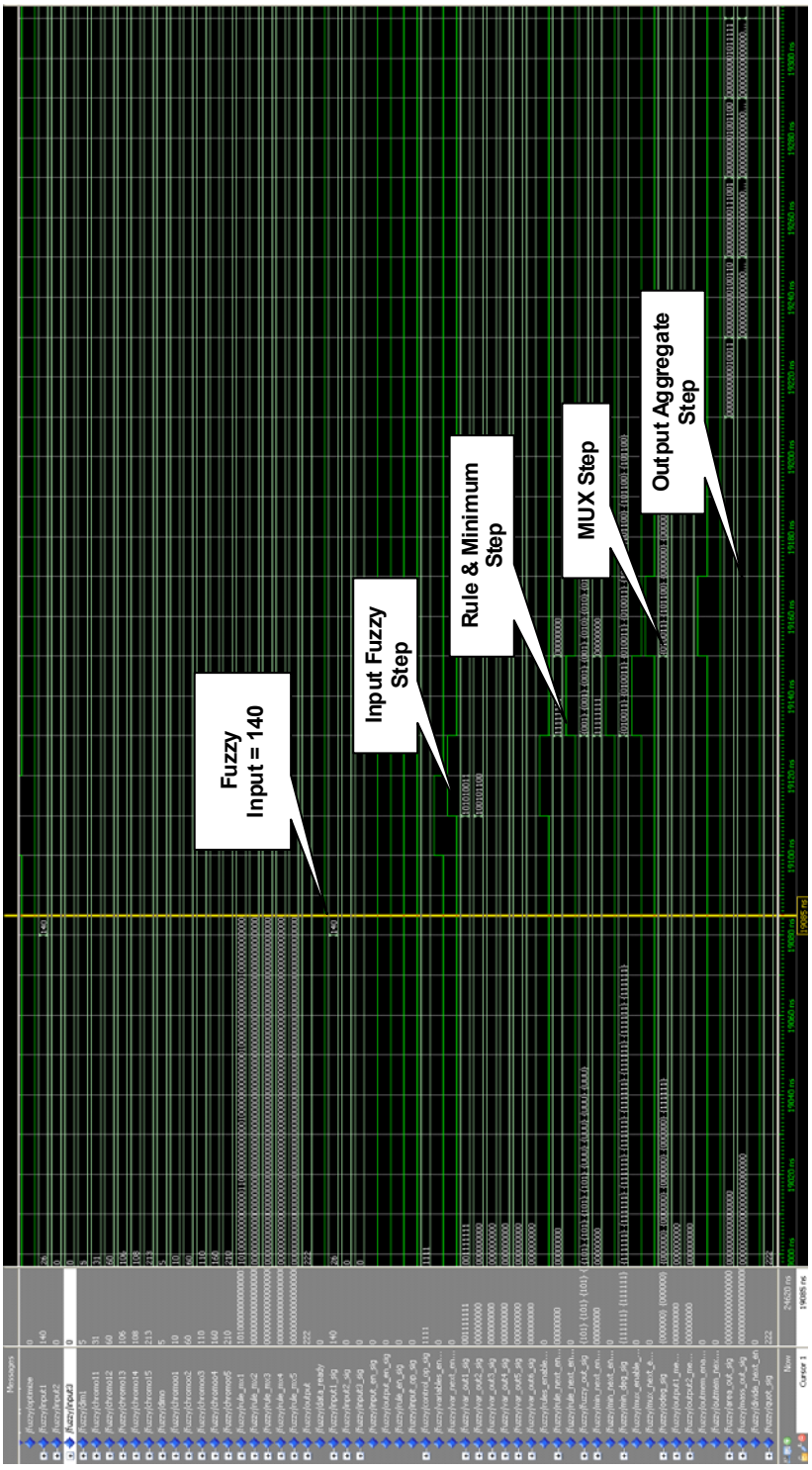

Figure 5.12 – Fuzzy Operation w/ Input = 140

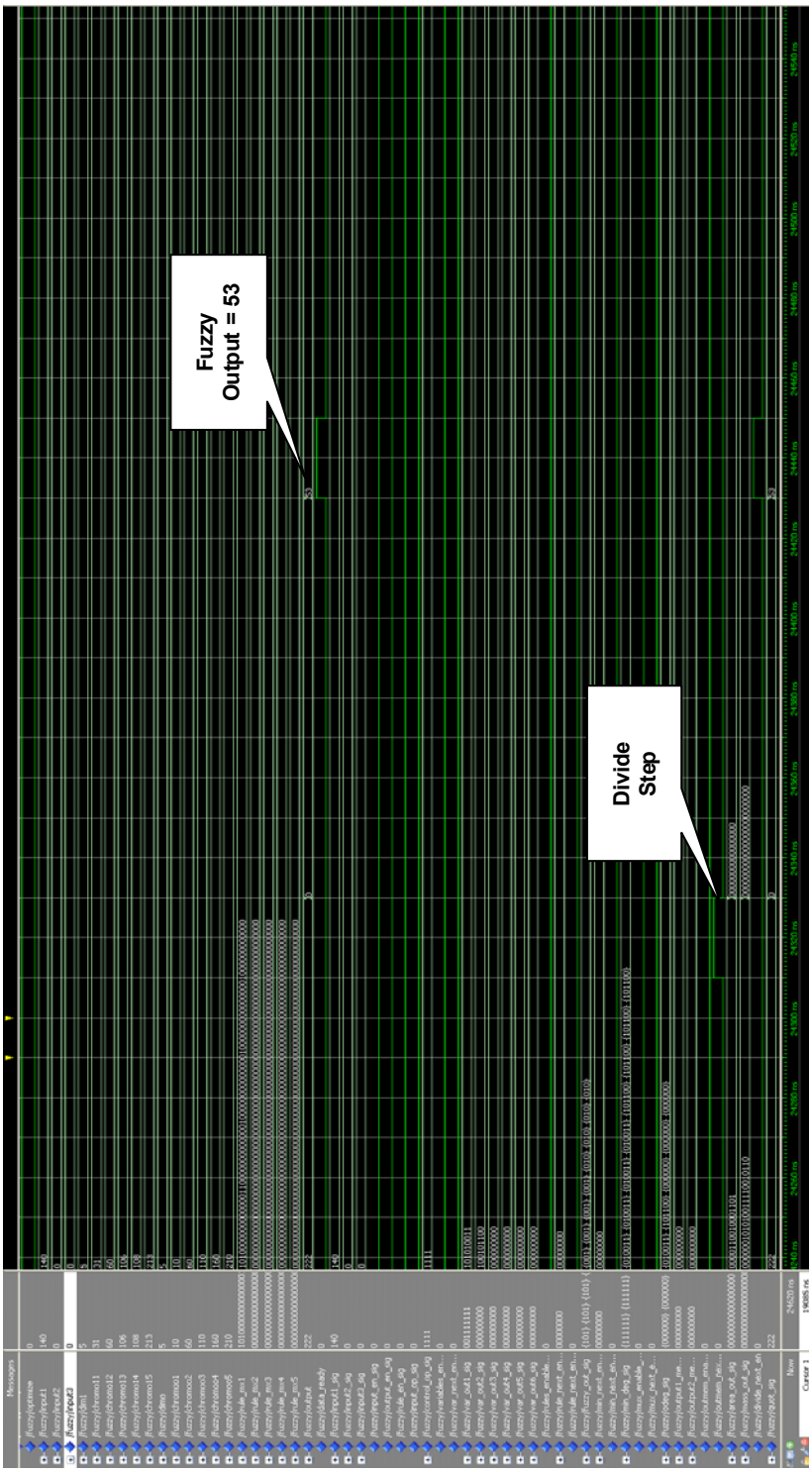

Figure 5.13 – Fuzzy Operation w/ Output = 53

The following simulations show the execution of each module in the controller and display the output response to each sample input. Figures 5.8, 5.10 and 5.12 present the sequence of events of the input fuzzy, rule matrix, minimum T-norm, MUX and output aggregate processes. Figures 5.9, 5.11 and 5.13 illustrate the division process and the final fuzzy output {222, 143, and 53}. When compared to software outputs {218, 149, 53} for the same input vector {20, 50, 140}, it was shown to have approximately 0% to 3% error. This error is directly linked to the margin of error from both the linear estimator in the fuzzy converter and the division module which substitute the mathematical division operator. The table below shows the actual "risk" value on 100% scale and the corresponding error.

Input 1 (BPM)	H/W Output	S/W Output	Actual Error (%)
20	101%	100%	1%
50	62%	65%	3%
140	17%	17%	0%
		Average	1.33%

The next set of simulation waveforms display the output of a two-input system. In this simulation model, we load the encoded chromosomes for two of the three inputs using 25 rules. The encoded fuzzy membership chromosome = {31, 60, 106, 108, 213, 8, 88, 112, 132, 241}. The corresponding fuzzy membership graphs are shown below.

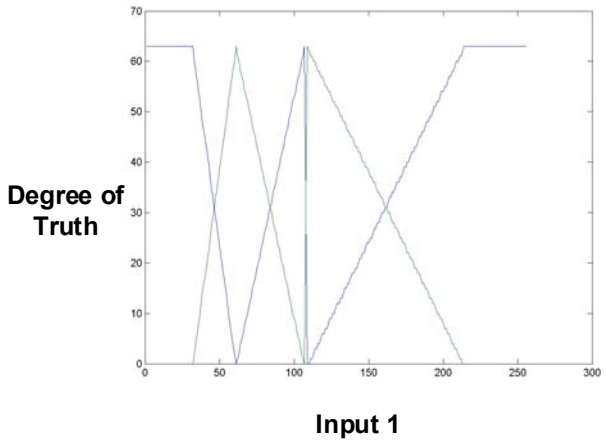

Figure 5.14 – Fuzzy Membership Graph – Input 1

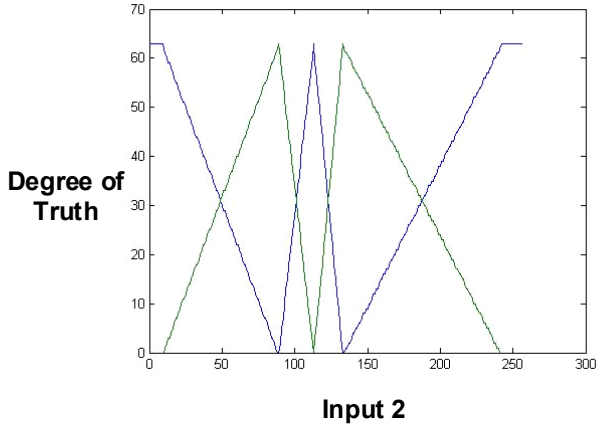

Figure 5.15 – Fuzzy Membership Graph – Input 2

The following diagram illustrates the rules matrix configured into the controller during optimization. The encoded chromosome = {5 5 4 3 3 5 4 3 2 2 4 3 2 2 1 3 2 2 1 1 2 2 1 1 1}.

Input 2

Input 1	1	2	3	4	5
1	5	5	4	3	3
2	5	4	3	2	2
3	4	3	2	2	1
4	3	2	2	1	1
5	2	2	1	1	1

Figure 5.16 – Two Input Rule Matrix

The output membership graph remains the same as the previous simulation.

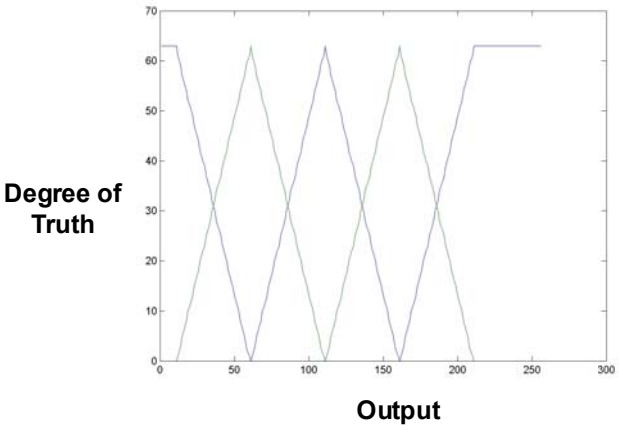

Figure 5.17 – Membership Graph for Output

The following simulation waveform displays the optimization process where the encoded chromosomes are loaded into the controller. The diagrams are cut short to display the critical parts of the waveform.

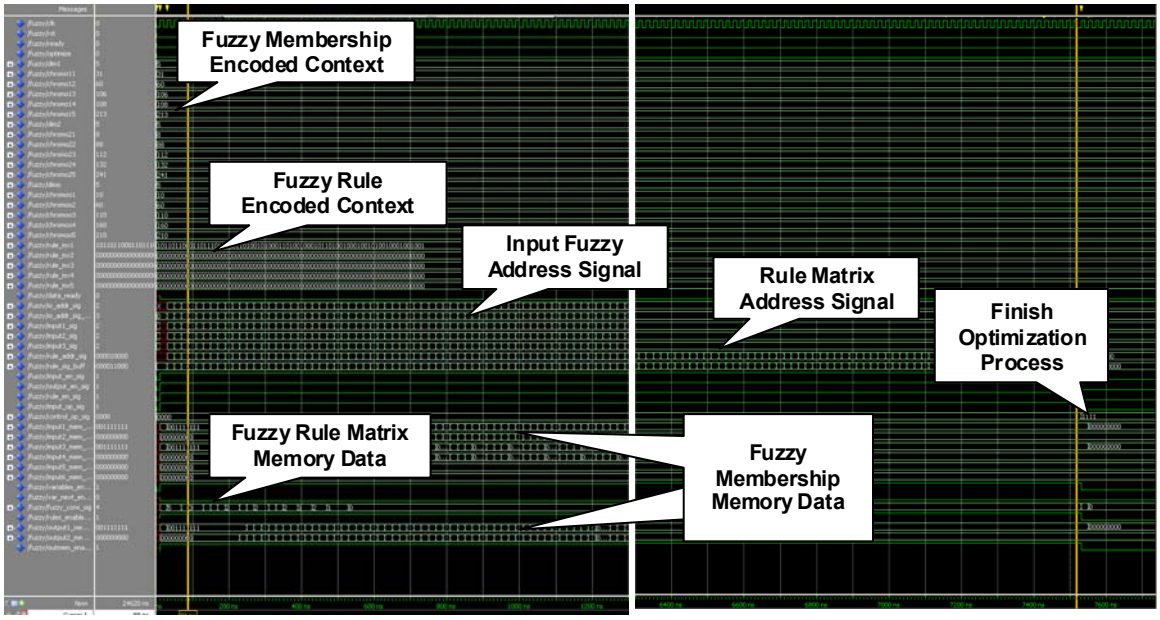

Figure 5.18 – Two Input Context Optimization Simulation

The remaining three waveforms display the input and output of fuzzy iterations selected for testing. The sample inputs for Input 1 = {20, 50, 140} and Input 2 = {5, 124, 160}.

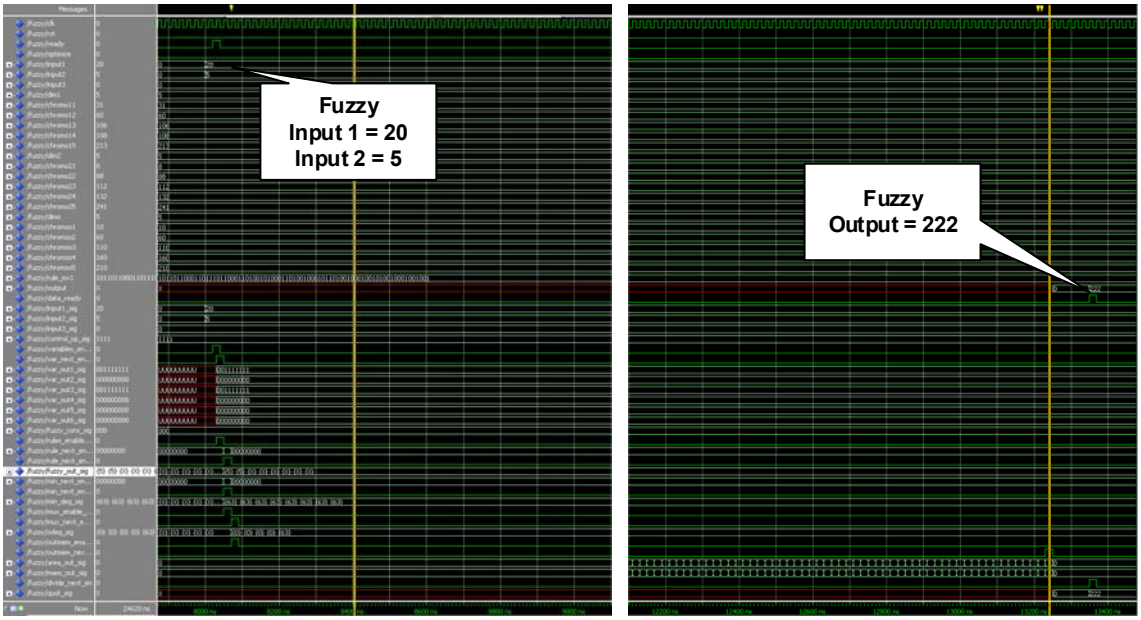

Figure 5.19 – Fuzzy Operation w/ Inputs {20, 5} – Output {222}

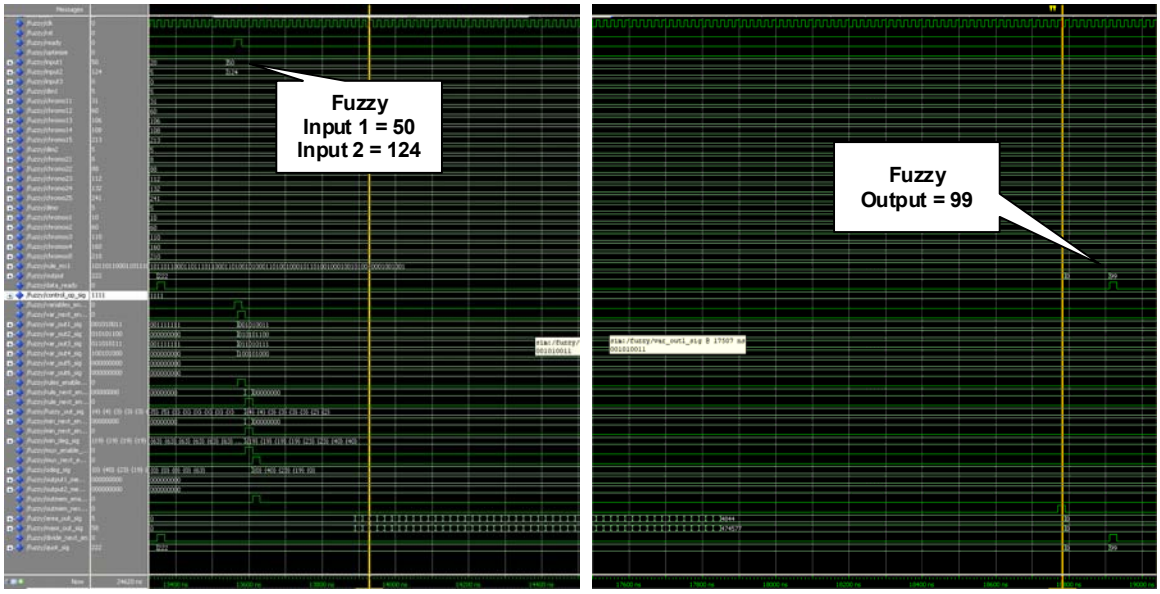

Figure 5.20 – Fuzzy Operation w/ Inputs {50, 124} – Output {99}

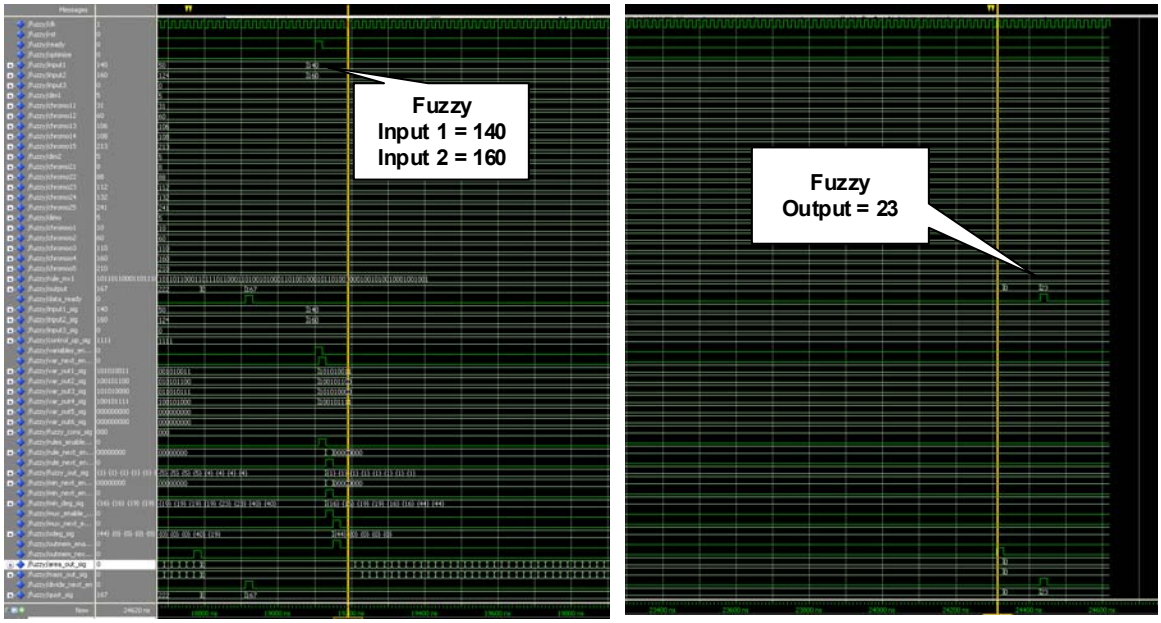

Figure 5.21 – Fuzzy Operation w/ Inputs {140, 160} – Output {23}

When we compared the hardware outputs {222, 99, and 23} to software outputs {218, 99, and 21} for the same inputs, it was shown to have approximately 0% to 2% error. As stated earlier, this error may be due to the substitute functions for division. The table below displays the actual "risk" values on a 100% scale and the corresponding error.

Input 1 (BPM)	Input 2 (BMI)	H/W Output	S/W Output	Actual Error (%)
20	1	101%	100%	1%
50	24.8	40%	40%	0%
140	32	2%	1%	1%
			Average	.66%

5.3 – SOFTWARE SIMULATIONS

There are two types of algorithms that have to be tested and simulated. The single input and multi-input evolutionary training algorithms are separately simulated using different specifications and stopping conditions. Both algorithms are structured similarly and are trained and tested with sample data that substantially represents the distribution of risk for sinus bradycardia. As stated earlier in Chapter 4, differences will exist between the two types of algorithms due to the variations in size of the solution space and output data points. These changes are made to increase effectiveness and efficiency. The multi-input algorithm has an assisted starting point for the multi-input search algorithm. Algorithm specifications and stopping conditions were also tested and modified to produce better solutions and make both algorithms more manageable. Important testing is done to carefully select the best combination of algorithm specifications.

Our training process will have three phases: training, validation and testing. Both single-input and multi-input training mechanisms undergo each phase. The training phase consists of searching for a set of solutions. The validation phase determines which of these solutions is most appropriate. The testing phase determines the error associated with the solution. Each phase has separate sample target datasets. The next sections will discuss algorithm specifications and the variable ranges for each specification. They will also introduce the target datasets and how they are arranged and assembled.

5.3.1 - Algorithm Specifications

The evolutionary probabilities and process specifications have a direct effect on the algorithm. We experimented with various parameter settings to generate the solutions with the optimal rating including:

- population size
- evolutionary generations
- tournament size
- tournament selection pressure levels
- crossover and mutation probability levels
- maturity rates and population proportion
- final stopping conditions

To help ensure that these settings are properly calibrated, similar specifications were identified in related papers and used as a reference point. According to several papers, each setting should be adjusted depending on the size of the solution space and method of training.

Table 5.1 – Algorithm Specifications Ranges

	Range
Population Size	50-500
Generations	100 - 1000
Tournament Size	10-50
Selection Pressure	0.6 - 0.9
Crossover Probability	0.6 - 0.9
Mutation Probability	0.05 - 0.2
Maturity Percentage	0.6 – 1.0
Maturity Rating	0.4 – 0.9
Maturity Ratio	0.8 - 0.98

The settings must be determined based on the fact that the solution space and the number of output data points can both be very large. Therefore, the table above includes wide ranges for each specification that were used when simulating the algorithm.

We verified our iterative program and algorithm specifications by performing several tests using the 2-dimensional Shubert function [27].

$$f(x_1, x_2) = \sum_{j=1}^{5} j \cdot \cos[(j+1)x_1 + j]$$

$$\bullet \sum_{j=1}^{5} j \cdot \cos[(j+1)x_2 + j]$$

where $\quad -10 \le x_i \le 10 \quad i = 1,2$

Equation 5.1 – Shubert Equation

The iterative algorithm will search for the coordinates (x1, x2) of a global minima found in the 2-dimensional plot. There are approximately 760 local minima and only 18 global minima = -186.73. The following diagram displays the Shubert function graph.

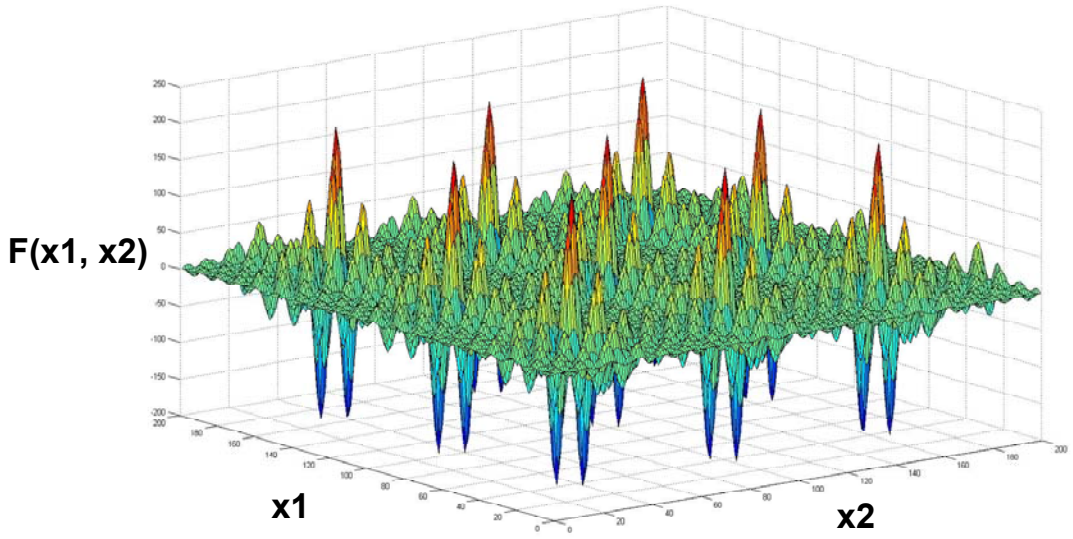

Figure 5.22 – 2-Dimensional Shubert Function

The top layer of our iterative algorithm searches for the x1 coordinate and the lower layer searches for the x2 coordinate. Therefore, the only aspect of the program that changes is the fitness function and length of each candidate. In place of the error-based fitness function which compares the fuzzy controller output with training data, we will use the Shubert equation. We were able to locate several coordinate points all with ***f(x1, x2)*** values of approximately -186. After establishing the ranges for algorithm specifications, the next step was to gather sample training data.

5.3.2 - Training Datasets

Training datasets are essential ingredients of the evolutionary optimization process. These datasets help configure the fuzzy membership graphs and rule propositions used in fuzzy operations. As stated in previous chapters, the number of possible output data points has a direct impact on the size of the training datasets. Each training dataset contains the input and output data pairs providing sample targets for the genetic algorithm. In our approach, we attempt to minimize the number of these points by performing multiple algorithms which include both single-input and multi-input based evolutionary programs.

In our simulations, we have three inputs: heart rate, age and body mass index. The output is the percentage risk of bradycardia. As a result, our training process will include three single-input training and one multi-input training algorithms. The three single-input training algorithms will search for solutions that can help provide initial guidance for the multi-input

training algorithm. The training dataset for each single-input algorithm provides the prevalence of sinus bradycardia based on each individual input.

For example, a population of patients is collected and tabulated using a graphical histogram for each chosen variable: heart rate, age or body mass index. Within each bin the number of patients diagnosed with the arrhythmia is collected and a percentage risk is assigned to that particular bin. The percentage of positively diagnosed patients is recorded as the prevalence of the disease for that particular group. A sample diagram is displayed below which illustrates how a population of diagnosed patients are categorized according to group section. The percentage of the population diagnosed with the arrhythmia is also shown for each section.

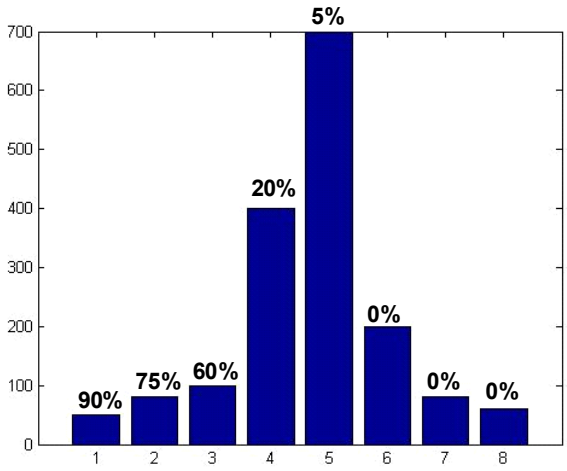

Figure 5.23 – Sample 1-Input Histogram Distribution of 1,670 Patients in 8 bins

The following diagram shows the prevalence of a disease based on the histogram shown above.

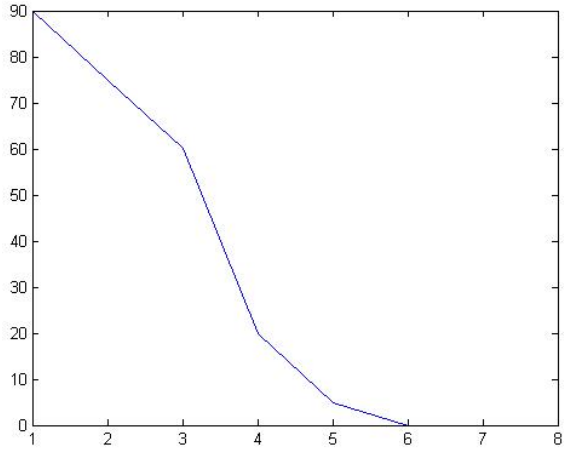

Figure 5.24 – Risk Distribution based on Sample 1-Input Histogram

The same process is done for multi-input algorithm training data sets where the histograms are categorized based on a combination of inputs ("heart rate and body mass index" or "heart rate and age"). The dimension of the histogram and risk distribution graphs depends on the number of inputs being considered. The following diagrams demonstrate the histogram and risk distribution for a disease considering two input medical data.

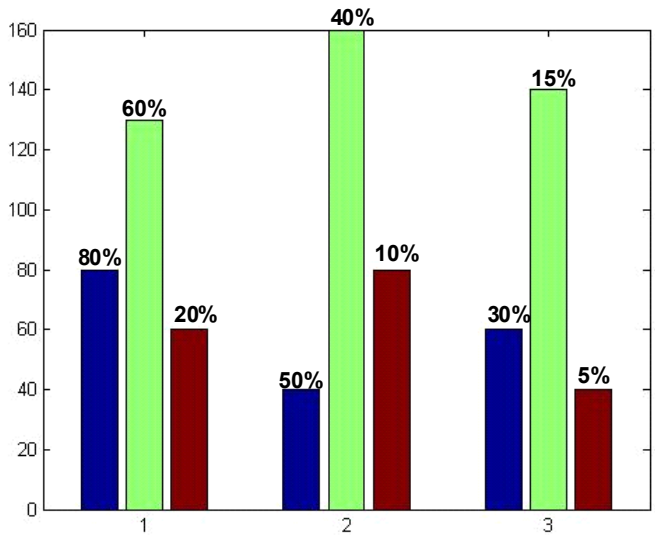

Figure 5.25 – Sample 2-Input Histogram Distribution of 790 Patients in 9 bins

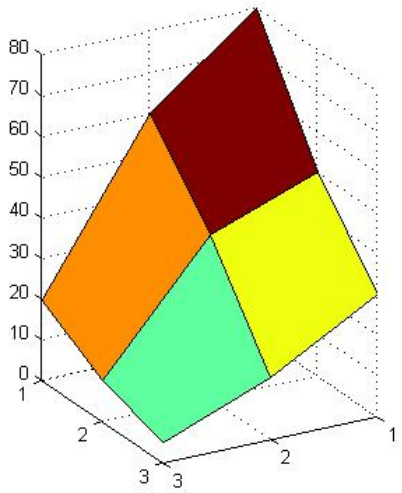

Figure 5.26 – Risk Distribution on Sample 2-Input Histogram

This information is gathered independently by epidemiological medical experts and clinical scientists who study the risk factors of diseases.

However after searching several databases for surveys on the prevalence of sinus bradycardia, we could not find any convenient statistical charts that concentrate on the combination of inputs required in this particular research project. To perform our own statistical analysis would take a significant amount of time and resources currently not available for our research.

To resolve this issue, we developed synthetic sample data, interpolated and extrapolated from substantive information provided by the American Medical Association and the World Health Organization. These groups have conducted fundamental research on sinus bradycardia and other cardiac arrhythmias. These references were used to formulate a set of sample data allowing for multi-algorithm based training. Since our medical risk controller is being designed to adapt and configure its internal parameters based on any given set of training data, the real-life practicality of the sample data is completely irrelevant. Our main goal is to prove that our multi-algorithm based training approach can accurately and effectively optimize fuzzy parameters according to the training data being provided. Extra target data samples are gathered for additional validation and testing. The ratio between the sizes of each dataset is 3:1:1. This ratio was taken from other papers which used similar techniques for training fuzzy classifiers.

5.3.2.1 – Sample Training Data The following charts show examples of various training data sets that were used to implement the training process.

Table 5.2 – Risk Distribution of Patients with Bradycardia based on Heart rate

Heart rate (BPM)	Prevalence of Bradycardia
0	100
40	83
60	47
100	24
160	15
180	10
200	5
220	0
240	0
255	0

Table 5.3 – Validation & Testing Data for Heart Rate

Heart rate (BPM)	Prevalence of Bradycardia
20	100
30	95
50	65
70	40
140	18
210	0

Table 5.4 – Risk Distribution of Patients with Bradycardia based on Age

Age (years)	Prevalence of Bradycardia
0	60
1	50
2	40
5	30
10	20
15	10
25	10
35	20
40	30
50	40
65	50
90	70
100	80
127	90

Table 5.5 – Validation & Testing Data for Age

Age (years)	Prevalence of Bradycardia
4	33
7	27
12	15
18	5
34	20
45	35
70	55
117	85

Table 5.6 – Risk Distribution of Patients with Bradycardia based on BMI

Body-Mass Index (BPM)	Prevalence of Bradycardia
0	45
5	40
10	30
15	25
20	20
25	20
30	25
35	30
45	40
50	45

Table 5.7 – Validation & Testing Data for BMI

Body-Mass Index (BPM)	Prevalence of Bradycardia
17	24
23	10
28	23
33	27
40	35
48	43

Table 5.8 – Risk Distribution of Patients with Bradycardia based on HR & BMI

HR \ BMI	0	5	15	20	25	35	45	50
0	100	100	100	100	100	100	100	100
40	80	60	20	10	0	30	60	80
60	60	50	20	0	0	20	40	60
100	30	20	0	0	0	0	10	20
160	0	0	0	0	0	0	0	0
180	0	0	0	0	0	0	0	0
220	0	0	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0

Table 5.9 – Validation and Testing Data for HR & BMI

HR	BMI	%
90	0	35
190	0	0
20	10	80
70	10	25
110	17	0
210	17	0
30	23	15
230	23	0
70	28	0
190	28	0
50	33	22
210	33	0
30	40	50
70	40	25
90	45	20
190	45	0
50	48	60
110	48	0
20	50	100
50	50	70

Table 5.10 – Risk Distribution of Patients with Bradycardia based on HR & Age

HR \ Age	0	2	5	15	35	50	65	100
0	100	100	100	100	100	100	100	100
40	90	80	60	40	30	50	60	80
60	70	60	40	20	10	30	40	60
100	60	50	30	10	0	20	30	50
160	40	30	10	0	0	0	10	30
180	0	0	0	0	0	0	0	0
220	0	0	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0

Table 5.11 – Validation & Testing Data for HR & Age

HR	Age	%
50	0	80
230	0	0
90	4	25
140	4	15
30	7	65
110	7	5
20	12	80
90	12	10
50	18	20
190	18	0
30	25	20
140	25	0
50	34	20
140	34	0
70	45	27
140	45	0
30	70	75
70	70	55
20	117	100
90	117	60

5.3.3 – Algorithm Simulation Results

The simulations performed for the single-input iterative algorithms showed great fitness results with minimal error. The genetic specifications chart for all single-input iterative algorithms is shown below.

Table 5.12 – Algorithm Specifications for Single-Input Iterative Algorithm

Genetic Specifications			
	Generations	400	
Membership Layer		**Rule Layer**	
Variable	**Range**	**Variable**	**Range**
Population Size	300	Population Size	150
Tournament Size	60	Tournament Size	30
Selection Pressure	0.9	Selection Pressure	0.9
Crossover Probability	0.8	Crossover Probability	0.8
Mutation Probability	0.1	Mutation Probability	0.1
Maturity Percentage	1.0	Maturity Percentage	1.0
Maturity Rating	0.9	Maturity Rating	0.9
Maturity Ratio	0.97	Maturity Ratio	0.97

The same output membership graph shown in Figure 5.17 was used for each training algorithm. We begin with heart-rate single-input iterative algorithm. The algorithm was simulated many times and several solutions were validated using the sample targets validation datasets. By manually adjusting the number of fuzzy sets for each simulation, different solutions

with different characteristics were retrieved. Solutions for the heart rate prevalence training were derived within an average time of approximately 25 minutes. Two of the best solutions are available below.

Table 5.13 – Heart Rate Trained Solutions w/ Validation & Scaled Testing Error

	Membership Chromosome	Rules Chromosome	Fitness	Validation Fitness	Testing Error
Solution 1	{36, 45, 63, 110, 217}	{5, 4, 3, 2, 1}	254	215	1.9%
Solution 2	{32, 72, 120, 210}	{5, 2, 2, 1}	250	436	0.7%

After validating each solution option, we selected and tested the error for two shown above. Ultimately, Solution 2 was chosen as the solution candidate. The diagrams for the fitness trends, membership graphs and output data plot are shown below.

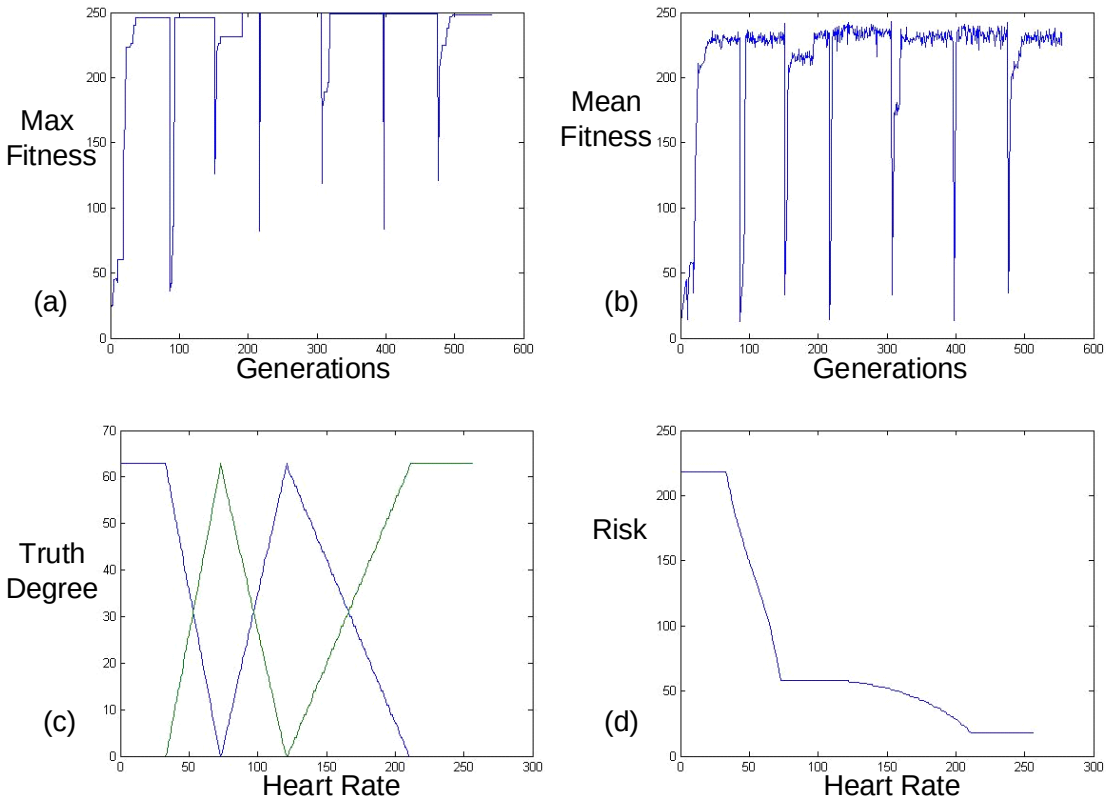

Figure 5.27 – HR Evolutionary Sequence for (a) Max & (b) Mean Fitness; (c) Membership Function for Heart Rate (d) Output Risk Plot for Heart Rate

This particular example demonstrates how reducing membership functions and rule propositions can have a positive influence on the evolutionary training algorithm. Figure 5.26(d) shows the risk of sinus bradycardia according to heart rate targets specified in the sample training datasets. This plot is the output of the genetically tuned fuzzy controller using sample inputs from 0 to 255. Next we will simulate the training for the age and body mass index risk factors.

The single-input iterative algorithm for body mass index produced extremely high rated solutions. The following chart shows several qualifying candidates that were validated and tested.

Table 5.14 – BMI Trained Solutions w/ Validation & Scaled Testing Error

	Membership Chromosome	Rules Chromosome	Fitness	Validation Fitness	Testing Error
Solution 1	{11, 12, 90, 129, 239}	{3, 3, 2, 2, 3}	408	219	2%
Solution 2	{8, 88, 112, 132, 241}	{3, 2, 2, 2, 3}	427	188	2%
Solution 3	{11, 92, 126, 244}	{3, 2, 2, 3}	408	279	1.7%

Solution 3 was selected for design implementation. The following figures simply illustrate the generational fitness trends, fuzzy membership graph and output plot of the solution.

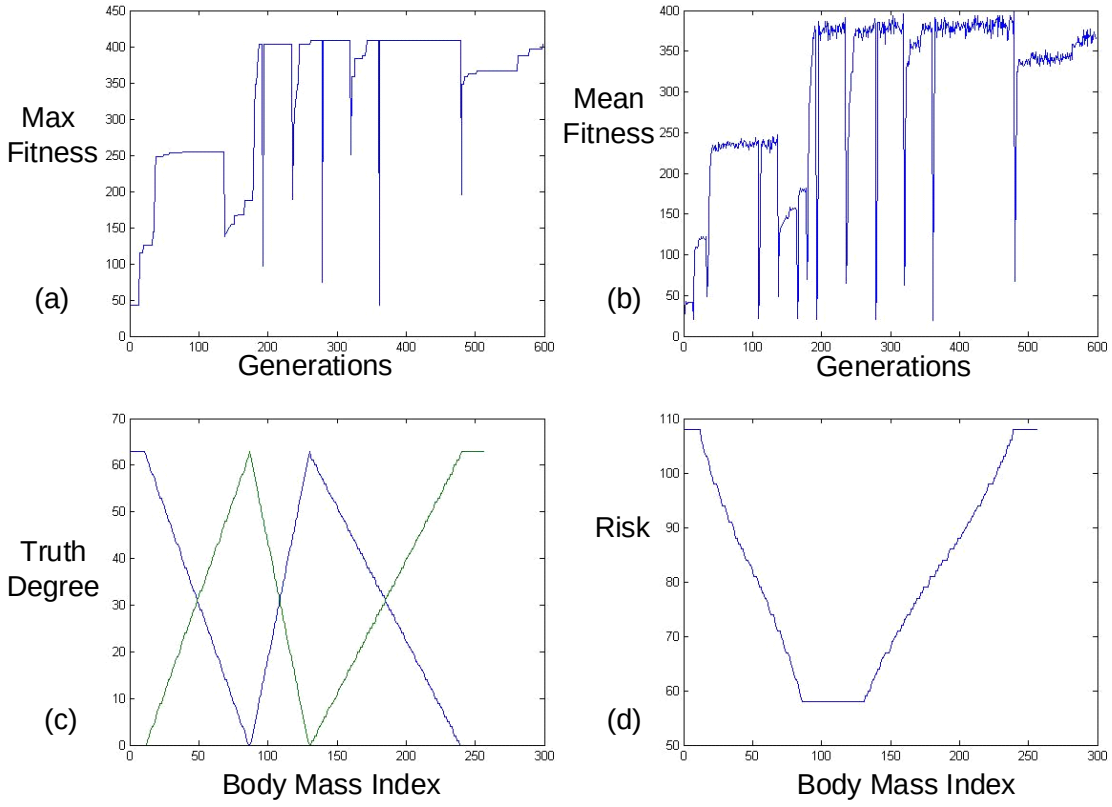

Figure 5.28 – BMI Evolutionary Sequence for (a) Max & (b) Mean Fitness; (c) Membership Function for Scaled BMI (d) Output Risk Plot for Scaled BMI

The body mass index Output Risk Plot is not fit to scale as in Figure 5.26(d). However, it does provide the risk levels for sinus bradycardia according to the figures that we developed for genetic training. This encoded solution also only contains 4 fuzzy sets and four rules. The final single input iterative algorithm is for age. The solutions with the best ratings are shown in the chart below.

Table 5.15 – Age Trained Solutions w/ Validation & Scaled Testing Error

	Membership Chromosome	Rules Chromosome	Fitness	Validation Fitness	Testing Error
Solution 1	{2, 16, 42, 180, 239}	{3, 2, 1, 4, 5}	86	171	2.5%
Solution 2	{3,14, 44, 116, 179}	{3, 2, 1, 3, 4}	64	174	2.5%

The first solution was selected to represent the parameters for the age single-input iterative algorithm. The following diagrams contain the fitness trend, membership graph and output data plot.

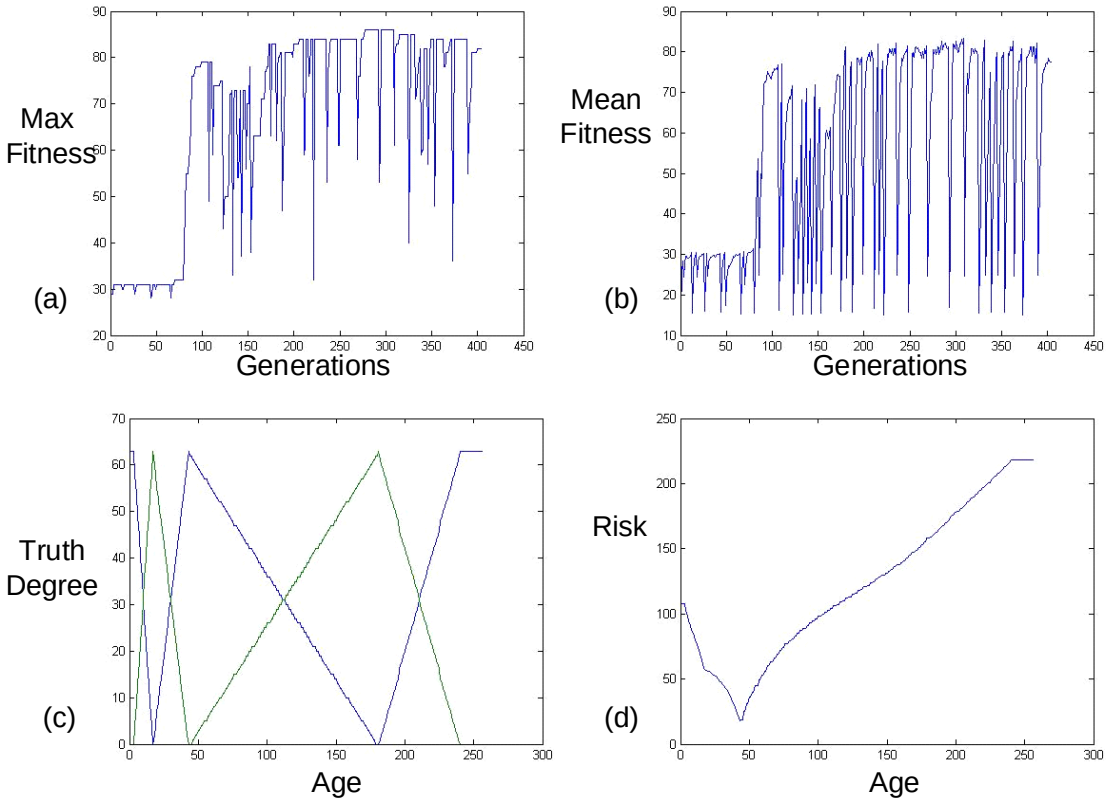

Figure 5.29 – Age Evolutionary Sequence for (a) Max & (b) Mean Fitness; (c) Membership Function for Scaled Age (d) Output Risk Plot for Scaled Age

These single-input training results will provide the necessary starting point for our multi-input training algorithms. The memberships of each solution will serve as the initial membership of multi-input algorithm in place of a membership chromosome selected at random. The encoded starting membership will vary depending on the inputs used by the multi-input algorithms. The next set of simulations will be based on the solution chromosomes from each individual single-input iterative algorithm.

5.4 – SOFTWARE TESTING RESULTS

This section presents the results of our multi-input algorithm training. As previously stated, the memberships of the single-input algorithm training will be used as starting points for this training. The training data used to perform the multi-input training is shown in Tables 5.8 – 5.11. For continuity, the algorithm specifications and output membership graph used for each multi-input training algorithm remained the same. Two separate training scenarios are simulated in this section for two-input systems which include heart rate-body mass index and heart rate-age.

Two solutions are shown in the diagrams below are optimized for the Heart-rate and Body Mass Index two-input training algorithm. Figures 5.30 (a – b) illustrate the evolutionary training cycle of the first fuzzy controller solution. The membership functions of each input are shown in Figures 5.30 (c-d). The rule matrix of the resulting solution and two angles of the output data plots are shown in Figures 5.31 and 5.32.

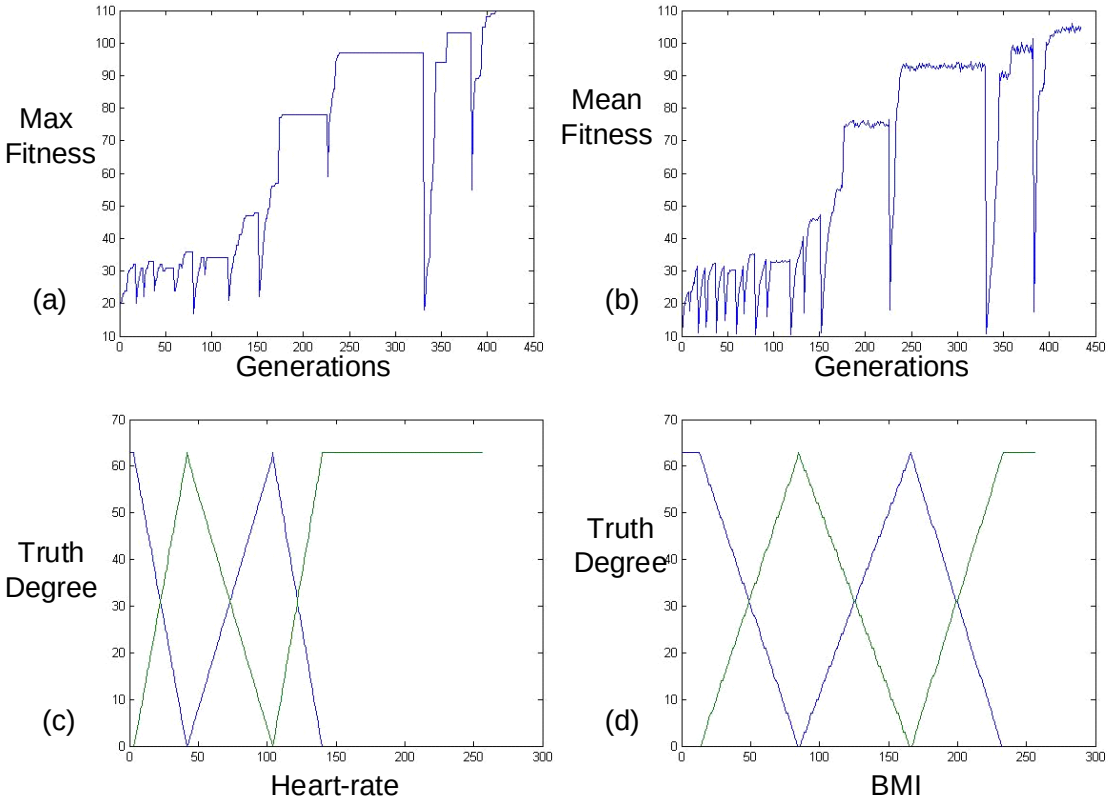

Figure 5.30 – HR-BMI Evolutionary Sequence for (a) Max & (b) Mean Fitness; Membership Function for (c) Scaled Heart rate and (d) Scaled BMI

Heart Rate \ BMI	1	2	3	4
1	5	5	5	5
2	4	1	1	2
3	2	1	1	1
4	1	1	1	1

Figure 5.31 – Heart-Rate vs. BMI Rule Matrix

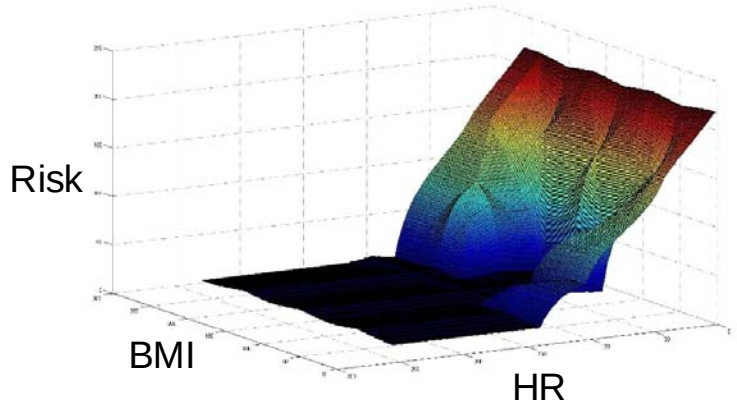

Figure 5.32 (a) – HR-BMI vs. Risk Output Data Plot Angle 1

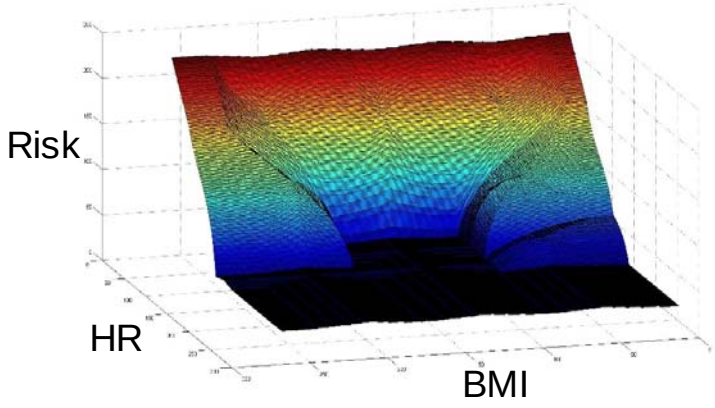

Figure 5.32 (b) – HR-BMI vs. Risk Output Data Plot Angle 2

Figures 5.33 (a – b) illustrate the evolutionary training cycle of the second optimized solution. The membership functions of each input are shown in Figures 5.33 (c-d). The rule matrix of the resulting solution and two angles of the output data plots are shown in Figures 5.34 and 5.35.

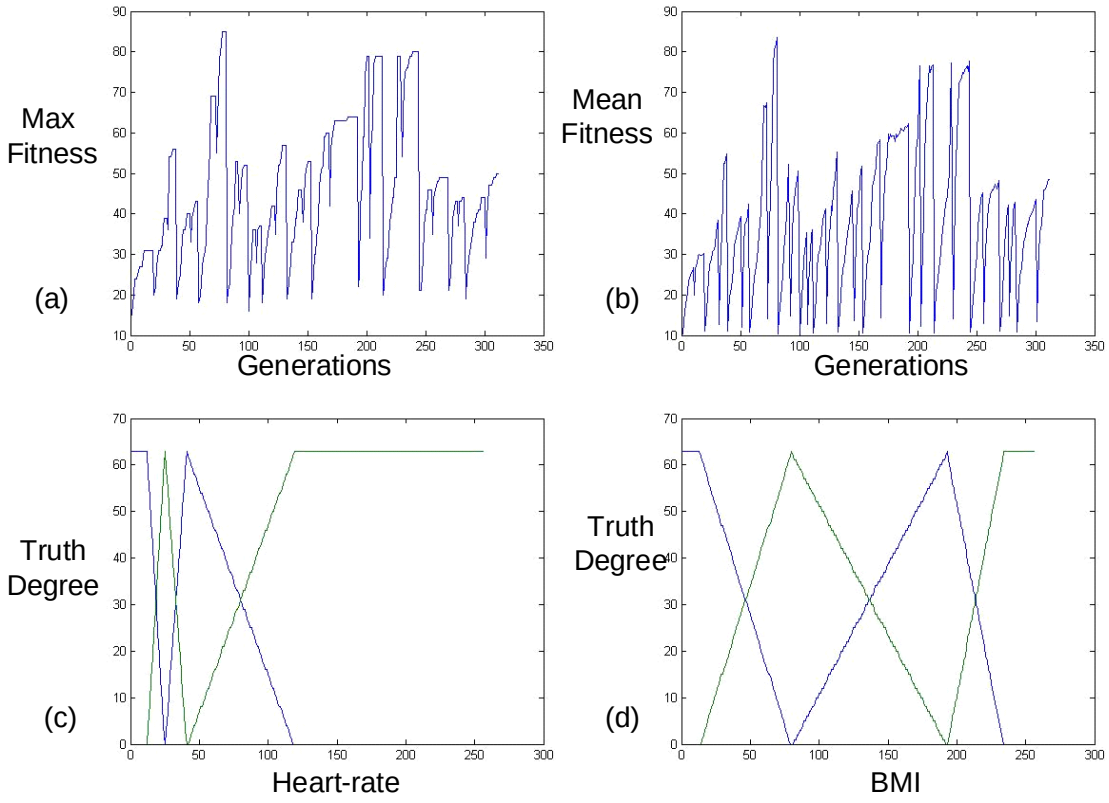

Figure 5.33 – HR-BMI Evolutionary Sequence for (a) Max & (b) Mean Fitness; Membership Function for (c) Scaled Heart rate and (d) Scaled BMI

The trained rule matrix of the resulting solution is shown in the following diagram.

Heart Rate \ BMI	***1***	***2***	***3***	***4***
1	5	5	5	5
2	4	2	3	4
3	4	1	2	4
4	1	1	1	1

Figure 5.34 – Heart-Rate vs. BMI Rule Matrix

The output plot for the trained input is also shown below from two separate angles.

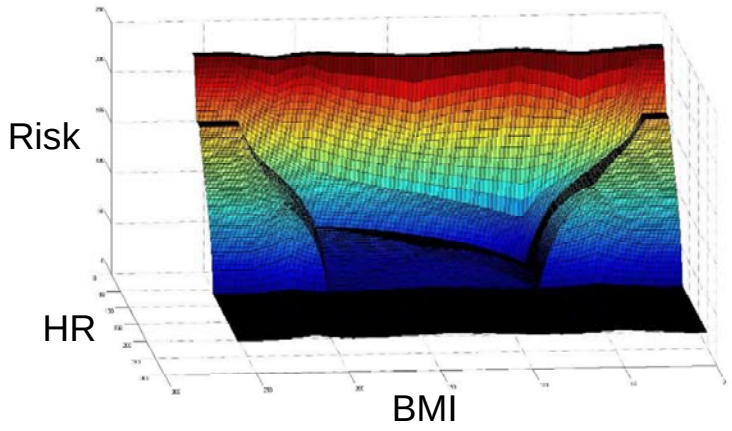

Figure 5.35 (a) – HR-BMI vs. Risk Output Data Plot Angle 1

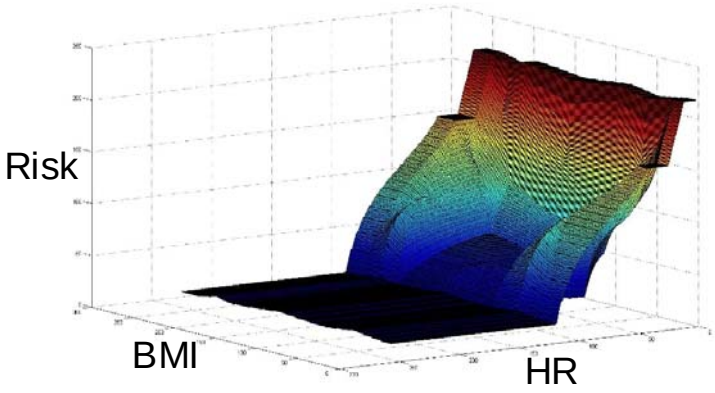

Figure 5.35 (b) – HR-BMI vs. Risk Output Data Plot Angle 2

The resulting fitness rating of the two optimized solutions are shown in the following chart.

Table 5.16 – HR-BMI Solution Fitness & Scaled Testing Error

	Fitness	Validation Fitness	Testing Error
Solution 1	110	41	11%
Solution 2	85	47	10%

The next two solutions detailed below are optimized for the Heart-rate and Age two-input training algorithm. Figures 5.36 (a – b) illustrate the evolutionary training cycle of an optimized solution. The membership functions of each input are shown in Figures 5.36 (c-d). The rule matrix of the resulting solution and two angles of the output data plots are shown in Figures 5.37 and 5.38.

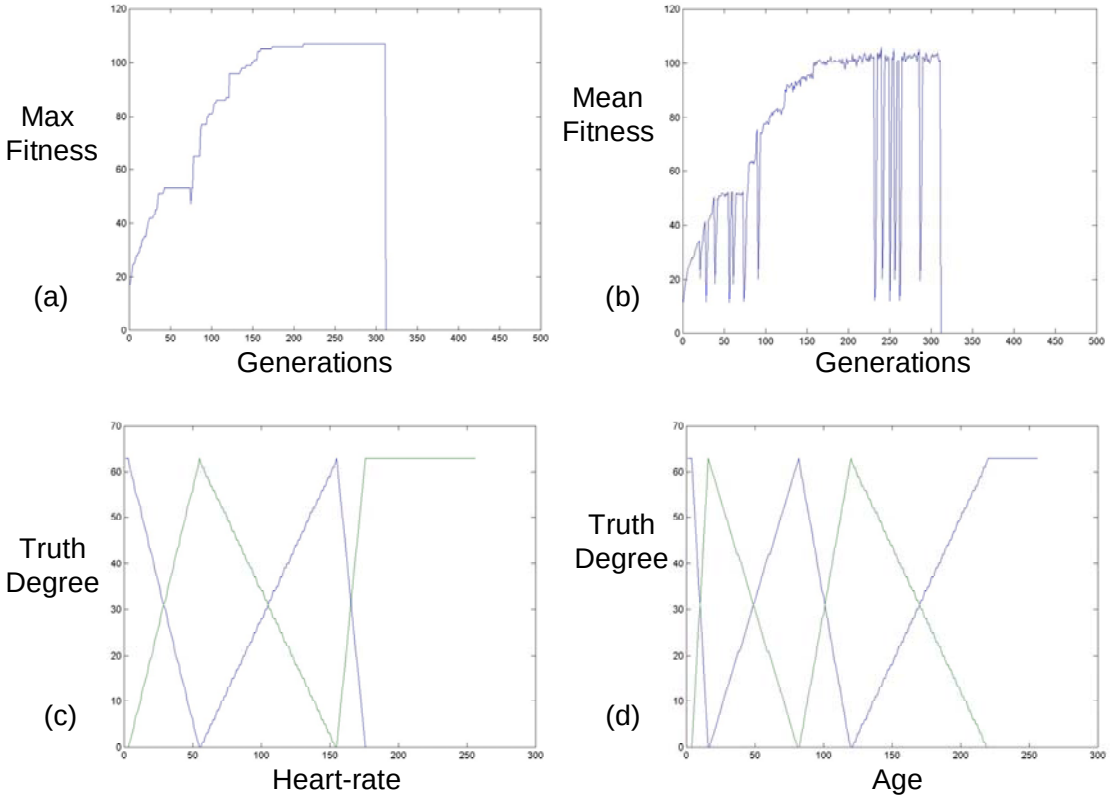

Figure 5.36 – HR-Age Evolutionary Sequence for (a) Max & (b) Mean Fitness; Membership Function for (c) Scaled Heart rate and (d) Scaled Age

Heart Rate \ Age	1	2	3	4	5
1	5	5	5	5	5
2	4	2	1	3	4
3	3	1	1	1	3
4	1	1	1	1	1

Figure 5.37 – Heart-Rate vs. Age Rule Matrix

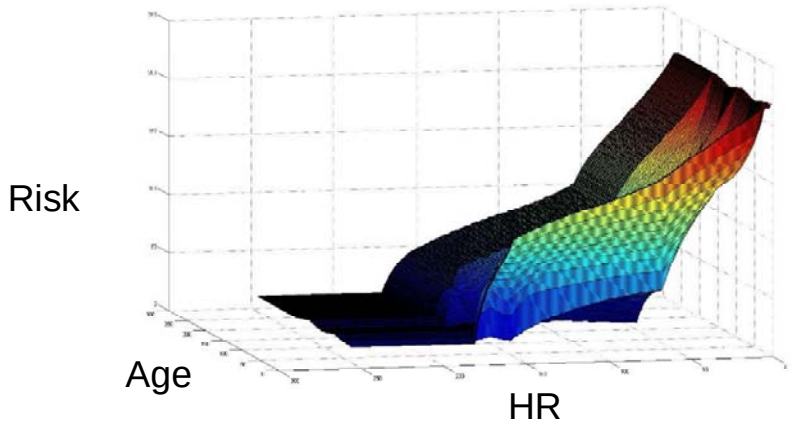

Figure 5.38 (a) – HR-Age vs. Risk Output Data Plot Angle 1

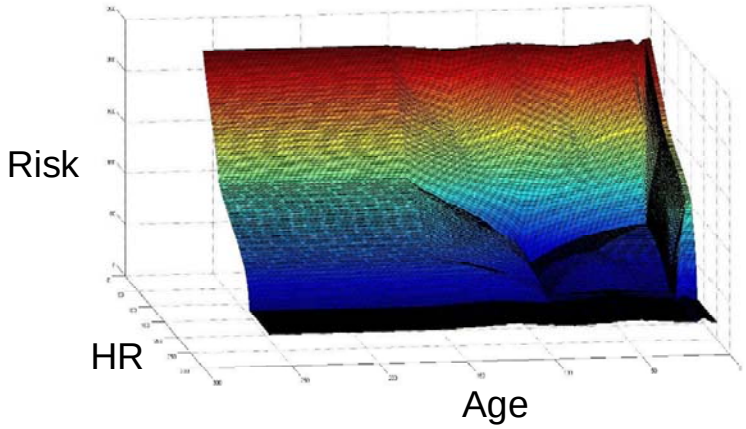

Figure 5.38 (b) – HR-Age vs. Risk Output Data Plot Angle 2

Figures 5.39 (a – b) illustrate the evolutionary training cycle of a second optimized solution. The membership functions of each input are shown in Figures 5.39 (c-d). The rule matrix of the resulting solution and two angles of the output data plots are shown in Figures 5.40 and 5.41.

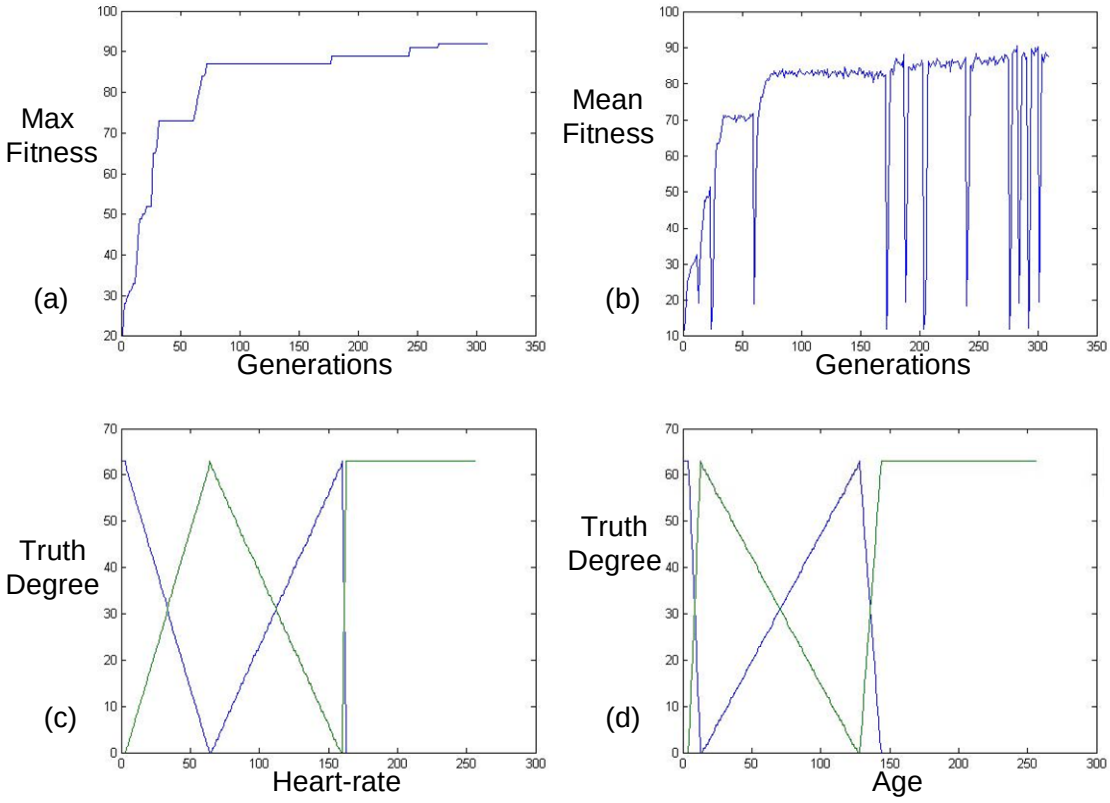

Figure 5.39 – HR-Age Evolutionary Sequence for (a) Max & (b) Mean Fitness; Membership Function for (c) Scaled Heart rate and (d) Scaled Age

The trained rule matrix of the resulting solution is shown in the following diagram.

Heart Rate \ Age	***1***	***2***	***3***	***4***
1	5	5	5	5
2	4	1	2	4
3	3	1	1	3
4	1	1	1	1

Figure 5.40 – Heart-Rate vs. Age Rule Matrix

The output plot for the trained input is also shown below from two separate angles.

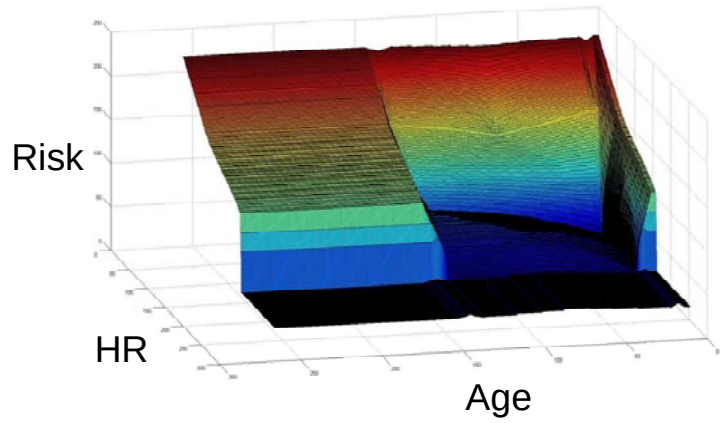

Figure 5.41 (a) – HR-Age vs. Risk Output Data Plot Angle 1

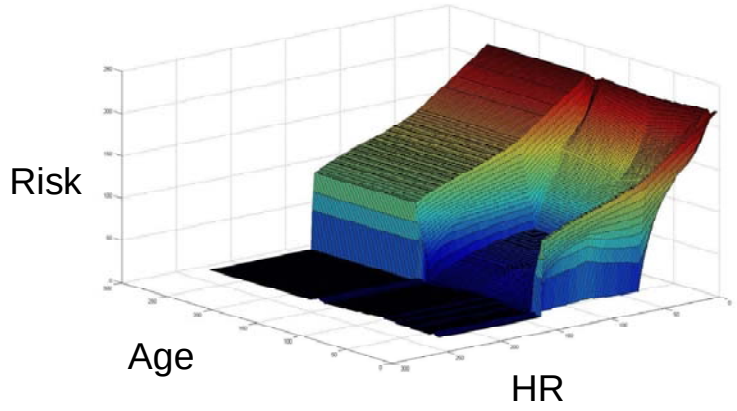

Figure 5.41 (b) – HR-Age vs. Risk Output Data Plot Angle 2

The resulting fitness ratings of the two optimized solutions are shown in the following chart.

Table 5.17 – HR-Age Solution Fitness & Scaled Testing Error

	Fitness	Validation Fitness	Testing Error
Solution 1	107	41	11%
Solution 2	92	38	12%

Although the fitness rating of each solution is relatively satisfactory, we recognized that more solutions needed to be trained, validated and tested for better results. In the next chapter we will discuss our results in detail and compare them with the results of the multi-input algorithm using the traditional approach. We will demonstrate that our methods prove to be more consistent and effective when compared to the traditional training approach. The fitness results of both approaches will be compared based on the same training data and algorithm specifications.

5.5 – HARDWARE PERFORMANCE ANALYSIS

The overall purpose of this research has two goals: improve hardware performance and device utilization of conventional fuzzy logic controllers and apply risk-based evaluation to fuzzy logic. This section will provide performance details about the fuzzy hardware design, which was implemented on a Xilinx Virtex-5 XC5VLX110T FPGA device.

Figure 5.42 – Xilinx Virtex-5 FPGA Evaluation Board

This information will be compared to other baseline studies referenced in the next chapter. Execution time was an important goal when designing our hardware system. Increasing parallelization and consolidating procedures helped us to shrink the execution time by a considerable amount.

The synthesis tool provided a timing summary on the input and output arrival delays and some guidance on the maximum frequency.

> Timing Summary (Speed Grade: -2)
>
> Minimum period: 27.880ns (Maximum Frequency: 35.868MHz)
>
> Minimum input arrival time before clock: 29.574ns
>
> Maximum output required time after clock: 2.826ns
>
> Maximum combinational path delay: No path found

The following chart describes each module and the cycles required for each stage. This chart enables us to find execution bottlenecks that can slow down the system. As stated in previous chapters, the timely execution of the

controller fuzzy operations is very vital to this research. The next chapter will dig deeper into these results.

Table 5.18 – Modular Timing Breakdown

Module Unit	Cycles
Controller	
- Load Membership	256
- Load Rules	125
Fuzzy Converter	1
Rule Converter	1
Input Fuzzify	
- Write Membership	1
- Read Membership	1
Rule Matrix	
- Write Rule Proposition	1
- Read Rule Proposition	1
Minimum T-norm	1
MUX Unit	1
Output Aggregate	
- Write Membership	1
- Read Mass & Area	256
- Using 8 Parallel Units*	32*
Divide Unit	<= 5
Total Optimization Cycles	381
Total Execution Cycles	36* - 265

Another critical aspect of the design that was taken into consideration was the device area. The following chart gives the hardware design summary

which outlines the amount of logic and memory units configured in the design. Many aspects of this chart were pleasing since we showed a significant decrease in device utilization compared to other designs. These discoveries will be further discussed in the next chapter.

Table 5.19 – Device Utilization for Xilinx Virtex-5 XC5VLX110T

Design Summary	Used	Available	Percentage
Slice Logic Utilization			
No. of Slice Registers	516	69,120	1%
- Used as Flip Flops	404		
- Used as Latch-thru	112		
No. of Slice LUTs	7,701	69,120	11%
- Used as Logic	7,630		
- Used as Memory	48	17,920	1%
- Single Port RAM	48		
- Used as ex route-thru	23		
No. of route-thrus	160	138,240	
Slice Logic Distribution			
No. of Occupied Slices	3,101	17,280	17%
No. of LUT-FF Pairs	7,763		
- Unused FF	7,247	7,763	93%
- Unused LUT	62	7,763	1%
- Fully used	454	7,763	5%
- Unique control sets	66		
I/O Utilization			
No. of Bonded IOBs	584	640	91%
Specific Features			
No. of Block RAMs	4	148	2%
- No. 18k Block RAMs	8		
- Total Memory (KB)	144	5,328	2%
No. of BUFG	1	32	3%
No. of DSP48Es	3	64	4%

The next chart breaks down the logic and memory utilization according to design module. From this chart we can some trade-offs that can made to make the design even more efficient and effective.

Table 5.20 – Modular Device Utilization Summary

Module	Slices	Slice Register	LUTs	LUTRAM	Block RAMs	DSPs	BUFG
Fuzzy	598 / 3741	129 / 516	310 / 7701	0 / 48	0 / 4	0 / 3	1 / 1
Control Unit	19 / 19	22 / 22	31 / 31	0 / 0	0 / 0	0 / 0	0 / 0
Fuzzy Conv. 1	583 / 583	28 / 28	1454 / 1454	0 / 0	0 / 0	2 / 2	0 / 0
Fuzzy Conv. 2	590 / 590	28 / 28	1454 / 1454	0 / 0	0 / 0	0 / 0	0 / 0
Fuzzy Conv. 3	576 / 576	28 / 28	1454 / 1454	0 / 0	0 / 0	0 / 0	0 / 0
Fuzzy Out Conv.	589 / 589	28 / 28	1435 / 1435	0 / 0	0 / 0	0 / 0	0 / 0
Input Fuzzify	2 / 2	1 / 1	2 / 2	0 / 0	3 / 3	0 / 0	0 / 0
Min1	19 / 19	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
Min2	21 / 21	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
Min3	20 / 20	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
Min4	20 / 20	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
Min5	19 / 19	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
Min6	19 / 19	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
Min7	19 / 19	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
Min8	19 / 19	7 / 7	37 / 37	0 / 0	0 / 0	0 / 0	0 / 0
MUX	158 / 158	31 / 31	334 / 334	0 / 0	0 / 0	0 / 0	0 / 0
Rule1	30 / 30	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule2	30 / 30	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule3	31 / 31	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule4	33 / 33	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule5	32 / 32	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule6	35 / 35	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule7	35 / 35	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule8	34 / 34	4 / 4	73 / 73	6 / 6	0 / 0	0 / 0	0 / 0
Rule Conv.	86 / 86	3 / 3	136 / 136	0 / 0	0 / 0	0 / 0	0 / 0
Output Aggr.	75 / 75	80 / 80	109 / 109	0 / 0	1 / 1	1 / 1	0 / 0
Divide Unit	49 / 49	50 / 50	102 / 102	0 / 0	0 / 0	0 / 0	0 / 0

These three tables inform us of the critical goals that were met and other that were not. When comparing this research to other papers, the goal was to acquire the advantages of each design and eliminate disadvantages by compromising some trade-offs between the desire for low area consumption and high speed capability. The next chapter will provide more detail into the analysis of meeting our desired goals.

6.0 – DISCUSSION

The technical charts in the previous chapter provide us with many insights into the improvements that have been made over other related designs. As stated earlier, the goal was to build off what has been proposed and remove their respective disadvantages. The authors of several papers [12, 13, 15, 16, 22, and 34] which were referred earlier in this paper each provide insightful ideas on how to design a hardware-based fuzzy logic controller. Authors of [13 and 15] provide high-speed FPGA-based conventional models which are neither trainable nor flexible. The expert knowledge and fuzzy calculations are pre-stored in memory which eliminates the flexibility needed to properly apply a fuzzy controller to multiple complex problems. The design is perfect for simple applications; however, it would not be a practical solution for medical diagnostics. By building on the parallel network to include fuzzy parameter converters which can dynamically modify the memory contents, we were able to increase the possibility for training and programming the controller to multiple complex purposes.

The author of [12] provided thoughtful approaches in optimizing and reforming the fuzzy hardware system by limiting rule memory modules to active rules and implementing an innovative strategy for calculating the more effective center of area output. However, the added complexity and

memory requirements to implement the partial sums can make the process ineffective in some scenarios. As a result, we expanded the accumulation technique used to calculate aggregate mass and area to the entire graph which limited the need for added memory and design complexity. This approach also increased the possibility for parallelizing the accumulation process.

The main baseline projects referenced in this paper are sited in [16, 22 and 34]. Each paper provides innovate approaches to solving timing and area utilization issues associated with fuzzy logic hardware design. Authors of [16] attempted to use asynchronous successive function blocks. This method produced significant execution time reduction for highly complex systems with more than 3 inputs and at least 25 rules. Authors in [22] employed highly parallelized and high-performance fuzzy techniques to obtain major speed-up in fuzzy operations while keeping design area to a limit. The researchers of [34] combined ASIC and FPGA systems to implement a genetically trained context switchable system that can optimize and manipulate fuzzy rule propositions online. They also claimed major speed-up in their fuzzy designs and utilizing minimal area.

In our research, we demonstrated that the resource consumption of our design is more cost effective. The current design uses less logic elements than the design in [34] while maintaining the ability to manipulate both fuzzy membership functions and rule parameters using encoded context registers. This is achieved by limiting the number of rule matrix block RAMs to only the number of rules that are ever active during a single fuzzy operation. In their research, the fuzzification phase was removed and the ability to identify selected fuzzy sets and active rule propositions was

eliminated. As a result, a rule memory was included for each proposition in the rule matrix.

To provide additional resource reduction, our research showed that sequential optimization can be effective. Since the area of a fuzzy converter is considerably large, the four parallel converters can be replaced with one sequential converter. The comparison is shown below. Our proposed design with parallel and sequential optimization technique can reduce the number of utilized CLB Slices by an average of approximately 40%.

Table 6.1 – FPGA Device Utilization Comparison

		Proposed System	
Resource	**Existing System [34]**	**Parallel Optimization**	**Sequential Optimization**
Function Generators (LUTs)	8,722	7,701	3,296
CLB Slices	4,361	3,101	1,350
Dff-Latches	329	516	174

In [22], the number of logic cells is listed next to each module in their research. Since their design does not contain a context switchable element, conventional wisdom would mean that their design will utilize less logic cells. However, their system employs similar techniques and modules that are used in our system like the binary search method in the antecedent unit in [22] and the membership graph converters in our proposed design. The following chart compares module unit based on common tasks.

Table 6.2 – FPGA Logic Cell Occupation Comparison

** = more than one unit in design*

Phase	Existing System [22]		Proposed System	
	Module Unit	Cells	Module Unit	Cells
Control	Controller	475	Control	38
Fuzzification	Input Interface	434	Fuzzy Converter*	1152-1180
	Detection*	122	Input Fuzzify*	4
	Antecedent*	174		
Inference + Aggregate	Rule*	267	Rule Converter	172
			Rule*	60-70
			Minimum*	38-42
			MUX	316
			Fuzzy Output Converter	1178
Defuzzification	Defuzzify	368	Output Aggregate	150
	Output Interface	106	Divide	98
Total	[22]	5887	Parallel Optimization	7585
			Sequential Optimization	3930

The system in [22] provide higher number of input variables and rule propositions, however, our design uses more resources for context switchability and execution time reduction. The comparison between the individual module units in each design clearly gives the advantage to [22]. When comparing our fuzzy converter and to their antecedent unit, it is important to note two noticeable advantages which include a much faster 1-cycle degree calculation and the ability to perform this task only once in the fuzzy operation. Our design only includes 3 inputs, 5 fuzzy sets, 1 output and 125 rules while their design can contain up to 16 inputs, 15 fuzzy sets, 1 output and 256 rules. One method to reduce this disparity is to offer sequential optimization, which limits the number of fuzzy converters to 1

and enables any number of inputs, fuzzy sets and rule propositions to be manipulated sequentially. The following graph shows the difference in logic cell utilization normalized to [22] as well as the savings produced from our improvised approach.

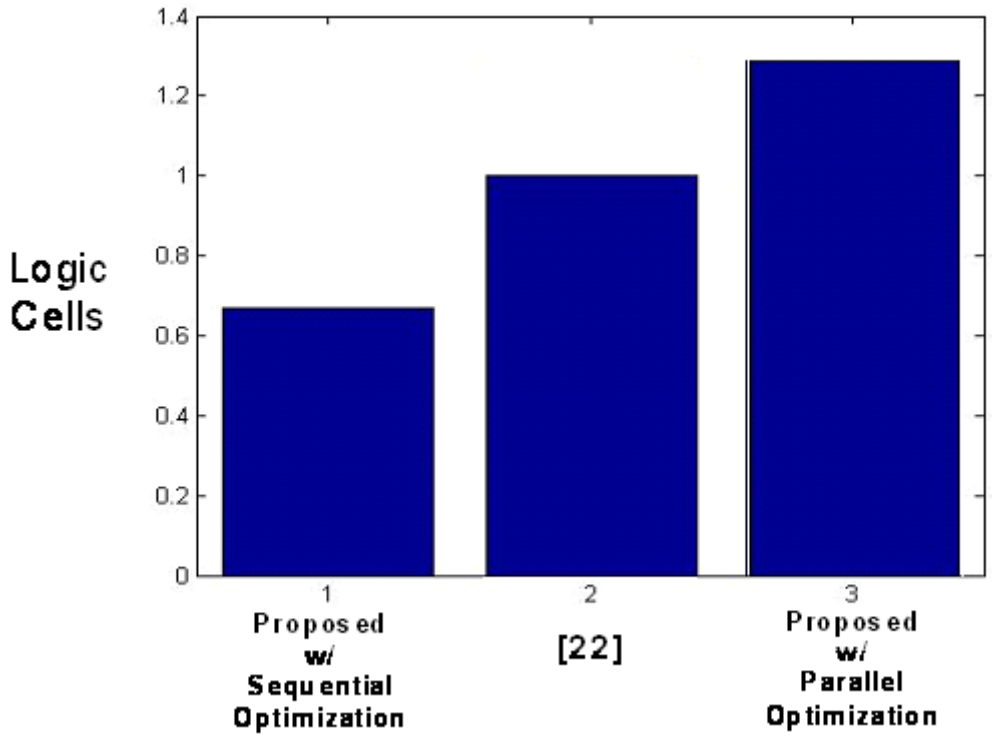

Figure 6.1 – Logic Cells Utilization

Our parallel optimization technique requires 7,585 logic cells, while the design proposed in [22] utilizes 5,887 logic cells. Eliminating the parallelization of our optimization reduces our resource utilization to 3,930 logic cells. This offers a reduction in logic cell utilization of approximately 25%.

In order to lower execution time, our design places a heavy emphasis on the use of a hybrid setup which uses both logic and memory blocks to

calculate, load and store data prior to fuzzy operations being executed. The goal is to have faster fuzzy operations occur without sacrificing losing flexibility or dynamicity. Authors in [15, 16, 22, 34] claim very high speeds using either millions of fuzzy logic inference per second (MFLIPS), cycles per iteration or seconds per iteration. All three papers opted to use center of gravity method for defuzzification to obtain faster speeds. Our goal is to show that center of area can be a viable and time efficient option in real time fuzzy controller applications. In our design, we used a single output aggregate unit to calculate the mass and area of the aggregate graph. According to timing breakdown figure, displayed in the previous chapter a system with a single output aggregate unit requires 265 cycles per fuzzy iteration or 6.625*u*s per iteration on a 40MHz clock. By parallelizing this phase using 8 output aggregate units the time can be dramatically improved. The risk of increasing the resource utilization can be alleviated by simultaneously de-parallelizing the optimization phase. The graph below compares seconds per iteration normalized to our proposed single output aggregate design and is shown below with [16, 22].

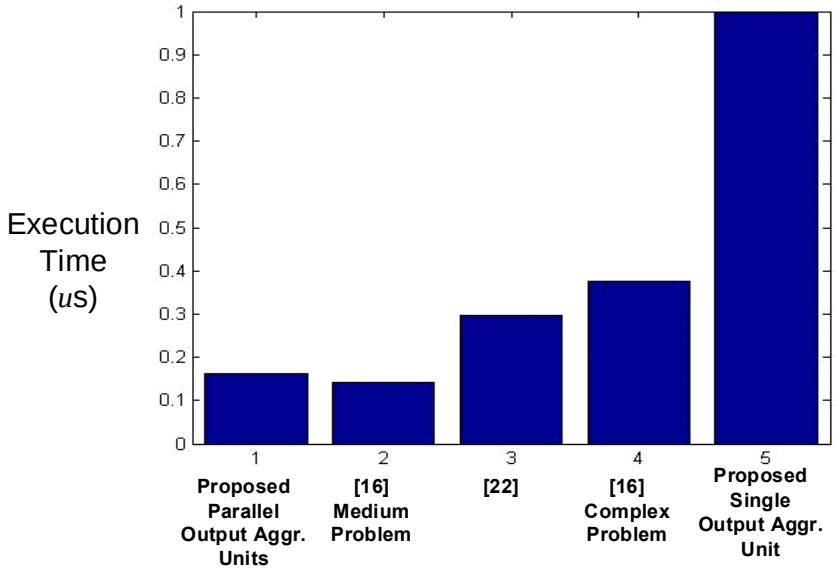

Figure 6.2 – Execution Time Comparison

Although the separate systems have different characteristics in terms of number of inputs, number of fuzzy sets, number of outputs and number of rules, our structural model certifies that the execution time will not be affected by these varying characteristics provided that the universe of discourse for the output membership graph is 256 elements. Our proposed output aggregate unit operates based on the number of elements in the universe of discourse. Outside the division phase, the system can be fully pipelined and parallelized using up to 256 units and requiring 1 clock cycle.

Other techniques were used to enhance operational speed and fuzzy context flexibility. Both the fuzzifier and inference calculations in [22] were replaced with memory read operations pre-loaded with data based on context memory. We expanded the context registers implemented in [34] to include fuzzy membership graphs in addition to the rule set matrix. Fuzzy converters employed the binary search algorithm in [22] to determine the

membership memory while rule converters utilize the decoding method in [34] to determine rule memory. This technique contributed to both functional and operational improvements.

This section provides area and execution time comparisons to closely related systems developed by several other researchers. We showed how by parallelizing the output aggregate units and de-parallelizing the optimization fuzzy converters, we can improve both fuzzy inference timing and remain area efficient. The advantages presented by each baseline paper was replicated and improved upon to enhance our proposed design while removing some disadvantages each project possessed. Much work can still be done to improve on the current design.

Next we will discuss some advantages gained from our proposed training approach. The following diagrams compare the fitness ratings and RMSE testing errors for solutions derived using both approaches. The graph compares the training results for two-input algorithms for heart rate-body-mass index and heart rate-age. In Figure 6.3, we demonstrate how our proposed method provides greater consistency, increased fitness and lower error for both two-input algorithms. We also use a variety of input dimensions and rule matrix array sizes to illustrate diverse results.

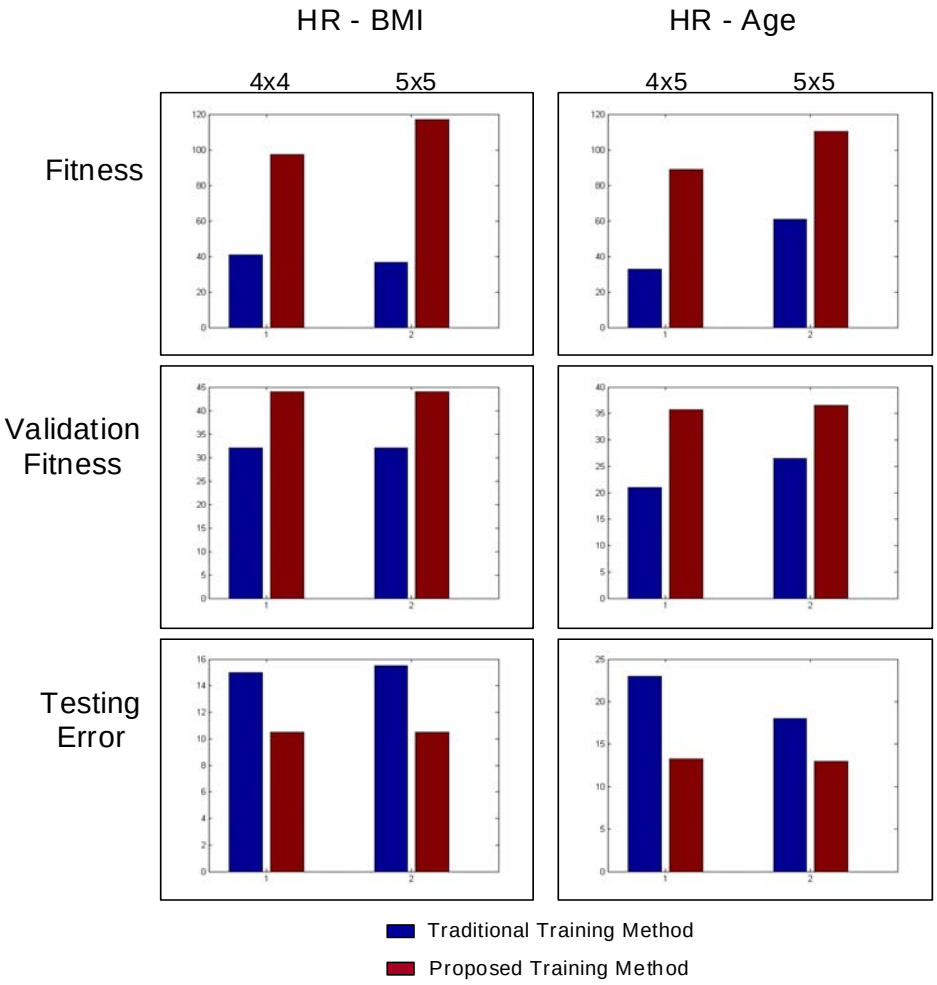

Figure 6.3 – Training Result Comparison

The above diagram also illustrates how our training approach can produce better results with the limited training samples within the same number of generations. The main motivation behind this training approach was driven by the lack of available medical samples that could substantially represent the prevalence and risk levels for multiple input combinations. This technique also allows for the effective training of a wide range of complex cardiovascular diseases where the solution space containing all possible rules and membership functions is considerably large. By searching

for solutions which portray the prevalence and contribution of each input risk factor, we can provide better initial assistance to the multi-input search algorithm. Obviously, a major concern is that the algorithms did not produce solutions with better accuracy; however this can be attributed to the need for more training. We are also confident that solutions with better accuracy can be developed without the restrictions we placed on the maximum generation count and population sizes to save time. The next section will describe future work for this project.

6.1 – FUTURE WORK

The system being implemented in this system meant to serve as a starting point to developing hardware-based medical risk evaluators using genetically tuned fuzzy logic parameters. Many aspects of this design can be augmented to improve output results and system flexibility. The most obvious change is the use of actual existing datasets for genetic training and testing. Due to lack of time and resources, medical surveys and reports could neither be found nor conducted that had the specific combination of variables represented in our simulation. The second obvious improvement or advancement that can be made is to expand the number of inputs and outputs of the system to cover larger more complex diseases. The possibility of tracking the trends of patient risk on cost effective platforms can be a major catalyst in both preventive medicine and cost control.

In the future, the membership graphs can be encoded using Gaussian bell curves or non-linked triangular fuzzy sets. By not limiting the graphs to linked triangular functions, genetic training programs can produce better results while the Gaussian bell curves are easier functions to calculate in the fuzzy converter module. Increasing the rule matrix size and fuzzy set overlap are two characteristics that can make the system more practical to complex control problems.

Training procedures have evolved by adding self-adjusting genetic probabilities based on maturity rate. This has been proven to increase the effectiveness of the solution search. Rule reduction methods are also popular tools for limiting the size of the matrix during the evolutionary cycle. Membership fuzzy set dimension reduction methods can also be included to help root out unnecessary fuzzy sets that may negatively impact the accuracy of the fuzzy controller outputs. Currently in our design, both processes are done manually.

Another future goal would be to replicate the ASIC-FPGA hybrid model in [27, 34] and compute genetic solutions online. The motivation would be to combine the PC and FPGA systems displayed in the following figure and develop a hybrid embedded system.

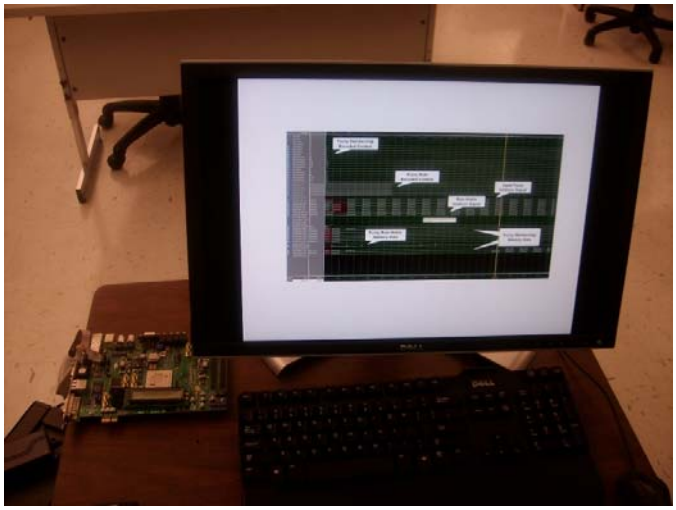

Figure 6.4 – Current Laboratory Setup

Software based iterative algorithm have powerful capabilities not shared with hardware-based models. However, if these genetic programs can be written in embedded C languages, our proposed fuzzy controller can be linked as a peripheral device taking advantage of the faster hardware based system. The fitness of population candidates can be determined using less execution time. Online training could allow new training data sample to be referenced on the fly. Also the microprocessor can be linked to various input and output peripheral devices contained in a user interface wrapper like keypads, monitors and telecommunications.

7.0 – CONCLUSION

7.1 – Summary

This research paper had two goals: design a more effective FPGA-based fuzzy logic controller and apply medical risk evaluation to that device. We were able to compile the design advantages introduced by several related papers and eliminate the disadvantages they unfortunately came with. By parallelizing, pipelining and consolidating the architecture, fuzzy context reconfigurability was another technique added to our controller design. We expanded on the ideas on papers and created a more flexible fuzzy controller at comparable speeds and area. This paper demonstrated how the controller can be expanded to include more applications, inputs and complexity without significantly added to the memory or execution time.

The lack of empirical formulas complicates the task of evaluating medical risk and makes it very difficult to program a computer device to perform the same task. The only expert-based evidence that is currently available is statistical data found in epidemiological surveys. Also this data is too often not available in the large quantity required for a computer to fully "learn" the technique of replicating the process. However, through the use of a different approach to genetic training, we showed that genetic-based

fuzzy logic solutions can provide better results for predicting medical risk using a limited number of medical samples compared to traditional methods in the same given time space. More simulations must be performed to search for better results than the ones displayed in our paper, but we are confident than the new approach being taken will produce more efficient solutions.

BIBLIOGRAPHY

[1] L.A. Zadeh, "Fuzzy Sets," *Inform. Control,* vol. 8, pp. 338-353, 1965.

[2] L.A. Zadeh, "Fuzzy Logic = Computing with Words," *IEEE Trans. on Fuzzy Syst.,* vol. 4, no. 2, pp. 103-111, 1996.

[3] E.H. Mamdani, "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis," *IEEE Trans. on Computers,* vol. C-26, no. 12, pp. 1182-1191, Dec. 1977.

[4] Wang, Li-Xin. *A Course in Fuzzy Systems and Control*. New Jersey: Prentice Hall, 1997.

[5] Gen, Mitsuo and Runwei Cheng. *Genetic Algorithms and Engineering Optimization*. New York: John Wiley & Sons, 2000.

[6] Zadeh, Lotfi A. "Is there a need for fuzzy logic," in *Proc. 27th North American Fuzzy Information Processing Society*, New York, NY, 2008, pp. 1-3.

[7] Langari, Reza. "Past, present and future of fuzzy control: A case for application of fuzzy logic in hierarchical control," in *Proc. 18th North American Fuzzy Information Processing Society*, New York, NY, 1999, pp. 760-765.

[8] Feng, Gang. "A Survey on Analysis and Design of Model-Based Fuzzy Control Systems," *IEEE Trans. Fuzzy Syst.,* vol. 14, no.5, pp.676-697, Oct. 2006.

[9] Jang, J.S.R. "ANFIS: Adaptive network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.,* vol. 23, pp. 665-685. May 1993.

[10] N. Golea, A. Golea and K. Benmahammed, "Fuzzy model reference adaptive control," *IEEE Trans. Fuzzy Syst.,* vol 10, no. 4, ppl 436-444. Aug. 2002.

[11] Ciftcioglu, Ozer. "Design enhancement by fuzzy logic in architecture," in *Proc. 12th IEEE Conf. on Fuzzy Syst.,* St. Louis, MO, 2003, pp. 79-84.

[12] Chiueh, Tzi-cker. "Optimization of Fuzzy Logic Implementation," in *Proc. 21st Int. Symp. On Multiple-Valued Logic,* Victoria, BC, 1991, pp. 348-355.

[13] Singh, Sameep and Rattan, Kuldip S. "Implementation of a Fuzzy Logic Controller on an FPGA using VHDL," in *Proc. 22nd North American Fuzzy Information Processing Society,* Chicago, IL, 2003, pp. 110-115.

[14] O'Brien, Amy J. "Fuzzy Inference Engine Provides Opportunity for Testbed," in *Proc. IEEE Int. Symposium on Circuits and Syst.,* Phoenix-Scottsdale, AZ, 2002, pp. 145-148.

[15] Hung, Donald. "Implementing a Fuzzy Inference Engine Using FPGA," in *Proc. 6th IEEE Int. ASIC Conf. and Exhibit,* Rochester, NY, 1993, pp. 349-352.

[16] Costa, Alessandra, De Gloria, Alessandro and Olivieri, Mauro. "Hardware Design of Asynchronous Fuzzy Controllers," *IEEE Trans. on Fuzzy Syst.,* vol. 4, no. 3, pp. 328-338, Aug. 1996.

[17] S. Chowdhury, A. Biswas and R. Chowdhury. "Design, Simulation and Testing of an Optimized Fuzzy Neural Network for Early Criticality Diagnosis," in *Proc. 1st Int. Conf. on Emerging Trends in Eng. and Techn.,* Maharashtra, India, 2008, pp. 665-670.

[18] Anuradha, B. and Reddy, V.C.Veera. "Cardiac arrhythmia classification using fuzzy classifiers," *Journal of Theoretical and Applied Information Techn.,* vol. 4, no. 4, pp. 353-359, 2008.

[19] M. Kundu, M. Nasipuri, and D.K. Basu. "A Knowledge-Based Approach to ECG Interpretation Using Fuzzy Logic," *IEEE Trans. on Syst., Man and Cybern. - Part B,* vol. 28, no. 2, pp. 237-243, Apr. 1998.

[20] Liang, Hong. "ECG Feature Elements Identification For Cardiologist Expert Diagnosis," in *Proc. 27th IEEE Conf. Engineering in Medicine and Biology,* Shanghai, China, 2005, pp. 3845-3848.

[21] Chandana, Sandeep and Mayorga, Rene V. "Fuzzy Rough & Rough Fuzzy Inference Engines: for Medical Applications,"in *Proc. Canadian Conference on Electrical and Computer Engineering,* Ottawa, ON, CA, 2006, pp. 567-570.

[22] Chowdhury, S.R. and Saha, H. "A High Performance FPGA-Based Fuzzy Processor Architecture for Medical Diagnosis," *IEEE Micro,* vol. 28, no. 5, pp. 38-52, Sept. 2008.

[23] Zong, W and Jiang, D. "Automated ECG Rhythm Analysis Using Fuzzy Reasoning," *Computers in Cardiology,* pp. 69-72, Sept 1998.

[24] R. Silipo, W. Zong and M. Berthold. "ECG Feature Relevance in a Fuzzy Arrhythmia Classifier," *Computers in Cardiology,* pp. 679-682, Sept. 1999.

[25] Stephen D. Scott, Ashok Samal and Sharad Seth. "HGA: A Hardware-Based Genetic Algorithm," in *Proc. 3rd Int. ACM Symposium on Field Prog. Gate Arrays,* St. Louis, MO, 1995, pp. 53-59.

[26] Tang, Wallace and Yip, Leslie. "Hardware Implementation of Genetic Algorithms Using FPGA," in *Proc. 47th IEEE Int. Midwest Symposium on Circuits and Syst.,* Hiroshima, Japan, 2004, pp. 549-552.

[27] P. Chen, R. Chen, Y. Chang, L. Shieh and H.A. Malki. "Hardware Implementation for a Genetic Algorithm," *IEEE Trans. on Instrumentation and Measurement,* vol. 57, no. 4, pp.699-705, Apr. 2008.

[28] Narayanan, Shruthi and Purdy, Carla. "Hardware Implementation of Genetic Algorithm Modules for Intelligent Systems," in *Proc. 48th IEEE Int. Midwest Symposium on Circuits and Syst.,* Cincinnati, OH, 2005, pp. 1733-1736.

[29] O. Cordon, F. Herrera, F. Hoffmann, F. Gomide and L. Magdalena. "Ten Years of Genetic Fuzzy Systems: Current Framework and New Trends," in *Proc. Joint 9th IFSA World Congress and 20th NAFIPS Int. Conf.,* Vancouver, BC, 2001, pp. 1241-1246.

[30] Castro, Pablo A.D. and Camargo, Heloisa A. "Learning and Optimization of Fuzzy Rule Base by Means of Self-Adaptive Genetic Algorithm," in *Proc. 13th IEEE Int. Conf. on Fuzzy Syst.,* Budapest, Hungary, 2004, pp. 1037-1042.

[31] Zuyuan Yang, Xiyue Huang, Hongfei Liu and Changcheng Xiang. "Multi-phase Traffic Signal Control for Isolated Intersections Based on Genetic Fuzzy Logic," in *Proc. 6th Congress on Intelligent Control and Automation,* Dalian, China, 2006, pp. 3391-3395.

[32] Chiou, Yu-Chiun and Lan, Lawrence W. "Adaptive Traffic Signal Control with Iterative Genetic Fuzzy Logic Controller (GFLC)," in *Proc. IEEE Int. Conf. on Networking, Sensing and Control,* Taipei, Taiwan, 2004, pp. 287-292.

[33] Chiou, Yu-Chiun and Lan, Lawrence W. "Genetic fuzzy logic controller: an iterative evolution algorithm with new encoding method," *Fuzzy Sets and Systems,* 152, pp.617-635, 2005.

[34] Q. Cao, M.H. Lim, J.H. Li, Y.S. Ong and W.L. Ng. "A Context Switchable Fuzzy Inference Chip," *IEEE Trans. on Fuzzy Syst.,* vol. 14, no. 4, pp. 552-567, Aug. 2006.

[35] T.W. Chua and W.W. Tan. "GA Optimisation of Non-Singleton Fuzzy Logic System for ECG Classification," in *Proc. IEEE Congress on Evolutionary Computation,* Singapore, 2007, pp. 1677-1684.

[36] Magdi B.M. Amien, Bo Cheng and Jiarui Lin. "Robust Techniques for Designing Remote Real-Time Arrhythmias Classification System," in *Proc. IEEE/NIH Life Science Systems and Applications Workshop,* Bethesda, MD, 2007, pp. 200-204.

[37] L. Pang, I. Tchoudovski, M.Braeclein, K. Egorouchkina, W. Kellermann and A. Bolz. "Real Time Heart Ischemia Detection in the

smart home care system," in *Proc. 27th IEEE Engineering in Medicine and Biology,* Shangai, China, 2005, pp. 3703-3706.

[38] H. Zhou, K.M. Hou, J. Ponsonnaille, L. Gineste and C. De Vaulx. "A Real-Time Continuous Cardiac Arrhythmias Detection System: RECAD," in *Proc. 27th IEEE Engineering in Medicine and Biology,* Shangai, China, 2005, pp. 875-881.

[39] Liu, Jihong and Liang, Deqin. "A Survey of FPGA-Based Hardware Implementation of ANNs," in *Proc.2nd IEEE Int. Conf. on Neural Networks and Brain*, Beijing, China, 2005, pp. 915-918.

[40] Galan, D., Jimenez, CJ., Barriga, A., Sanchez-Solano, S. "VHDL Package for Description of Fuzzy Logic Controllers," in *Proc. EURO-Design Automation Conference,* Brighton, UK, 1995, pp. 528-533.

[41] Jou, JM., Chen, P., Yang, S. "Design and Implementation of a Self-Adaptive Fuzzy Inference Engine," in *Proc. 4th IEEE Int. Joint Conf. on Fuzzy Systems and 2nd IEEE Int. Fuzzy Eng. Symp,* Yokohama, Japan, 1995, pp. 1633-1640.

[42] Homaifar, A. and McCormick, E. "Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms," in *IEEE Trans. on Fuzzy Syst.,* vol. 3, no.2, pp.129-139, May 1995.

[43] Park, S. and Lee-Kwang, H. "Designing Fuzzy Logic Controllers by Genetic Algorithms Considering Their Characteristics," in *Proc. of the 2000 Congress on Evolutionary Computation,* La Jolla, CA, 2000, pp.683-690.

[44] Virtex 5 Datasheets, http://www.xilinx.com

[45] American Heart Association, http://www.americanheart.org

[46] North American Association for the Study of Obesity, http://www.obesityresearch.org

[47] Wisconsin Women's Health Foundation, http://www.wwhf.org