Implementation of a Fuzzy Logic Controller via Cyclone II FPGA and Nios II Processor

by

Eric Michael Stauffer

Submitted in Partial Fulfillment of the Requirements

For the Degree of

Master of Science

In the

Electrical and Computer Engineering

Program

YOUNGSTOWN STATE UNIVERSITY

August, 2006

Implementation of a Fuzzy Logic Controller via Cyclone II FPGA and Nios II Processor

Eric Michael Stauffer

Signature: Eric Stauffer 7/26/06

Eric Michael Stauffer, Student Date

Approvals:

[illegible] 7/26/06

Dr. Faramarz Mossayebi, Thesis Advisor Date

[illegible] 7/26/2006

Dr. Jalal Jalali, Committee Member Date

P Munro 26-July-06

Dr. Philip Munro, Committee Member Date

[illegible] 1 August, 2006

Peter J. Kasvinsky, Dean of Graduate Studies Date

Abstract

Fuzzy control is an emerging control technique which allows an engineer to design a control system with only an operator's knowledge of the plant being controlled. FPGAs and soft processor cores allow engineers to rapidly prototype and modify systems such as controllers. The combination of these two emerging technologies is discussed in detail in this work. It is shown that the principals of fuzzy sets and fuzzy logic can be the basis of a control system implemented in FPGA hardware with a soft processor core. An Altera Cyclone II FPGA with a Nios II soft processor core is implemented to provide an empirical example of a fuzzy controlled system. The design process is detailed in a step-by-step fashion and a prototype is assembled and detailed. Experiments proving the validity of fuzzy control as a control scheme are designed and run and the results are provided. The experiments consists of a fuzzy system controlling the temperature of air inside of a confined space.

The results prove that fuzzy control is a viable control scheme and that the assertions about limited plant knowledge and ease of design are true. A temperature versus time plot is shown for the ambient temperature inside of a confined space in which a heating and exhaust system is controlled by a fuzzy-based controller.

Table of Contents

Chapter 1......1

1.1 Motivation and Background......1

1.2 History......3

1.3 Empirical Experiment......3

Chapter 2......5

2.1 Introduction to Fuzzy Sets......5

2.2 Membership Functions......8

Chapter 3......15

3.1 Introduction to Fuzzy Logic......15

3.2 Fuzzy Logic Operations......16

Chapter 4......20

4.1 Introduction to Fuzzy Control......20

4.2 Preliminary Design......22

4.3 Rule Base Creation......24

4.4 Input Processing......26

4.5 Logic Implementation and Rule Inference......27

4.6 Output Processing......29

Chapter 5......31

5.1 Motivation and Introduction......31

5.2 Controller Design, Step 1......32

5.3 Controller Design, Step 2....33

5.4 Controller Design, Step 3....33

5.5 Controller Design, Step 4....39

5.6 Controller Design, Step 6....40

5.7 Controller Design, Step 7....43

5.8 Controller Design, Step 8....43

5.9 Hardware....44

5.10 Nios II Processor....47

5.11 Experiment Results....49

5.12 Conclusions and Future Research....50

Appendices....53

Appendix A....54

A.1 Introduction....54

A.2 Notation and Operations....54

Appendix B....58

B.1 Introduction....58

B.2 Notation and Operations....58

Appendix C....61

C.1 Main Function....61

C.2 Declarations and Variables....67

C.3 Membership Functions....70

C.4 LCD and Buttons....76

C.5 Relay Functions........79

C.6 Temperature Readings........81

Appendix D........83

D.1 Introduction........83

D.2 Quartus II........83

D.3 SOPC Builder........86

D.4 Nios II IDE........87

References........88

List of Figures

Figure 2.1	Venn Diagram and Single Subset
Figure 2.2	Venn Diagram and Two Subsets
Figure 2.3	Venn Diagram Representing A Union B
Figure 2.4	Venn Diagram of A Intersection B
Figure 2.5	Venn Diagram Representation of Tall/Short Example
Figure 2.6	Venn Diagram Representing a Fuzzy Set Boundary
Figure 2.7	Venn Diagram Representation of Three Elements in a Fuzzy Set
Figure 2.8	Membership Function of Classical Set
Figure 2.9	Simple Linear Membership Function
Figure 2.10	Two Set Membership Function
Figure 2.11	Triangular Membership Function
Figure 2.12	Trapezoidal Membership Function
Figure 2.13	Two Trapezoidal Membership Functions
Figure 3.1	Membership Function Defining Membership in Two Subsets
Figure 4.1	Sample Fuzzy Controller Block Diagram
Figure 5.1	Hot Membership Function
Figure 5.2	Warm Membership Function
Figure 5.3	Neutral Membership Function
Figure 5.4	Cool Membership Function
Figure 5.5	Cold Membership Function

Figure 5.6 Temperature Membership Functions

Figure 5.7 Cooling Membership Function

Figure 5.8 Constant Temperature Membership Function

Figure 5.9 Warming Membership Function

Figure 5.10 Temperature Change Membership Functions

Figure 5.11 Altera DE2.

Figure 5.12 Carl's Electronics CK1619 Relay Board.

Figure 5.13 Complete System.

Figure 5.14 Micro-Controller Diagram.

Figure 5.15 System Response at Various Desired Temperatures.

List of Tables

Table 2.1	Membership Values for Figure 2.8
Table 2.2	Membership Values for Figure 2.9
Table 2.3	Membership Values for Figure 2.10
Table 3.1	Classical AND Function
Table 3.2	Fuzzy AND Function
Table 3.3	Classical OR Function
Table 3.4	Fuzzy OR Function
Table 3.5	Membership Values Derived From Figure 3.1
Table 4.1	Water Temperature Change Rule Matrix
Table 5.1	System Rule Base
Table B.1	Conjunctive Truth Table
Table B.2	Negation Truth Table
Table B.3	Implicative Truth Table
Table B.4	Equivalent Truth Table

List of Symbols

A, B, C, X, Y	Subsets of the Universe of Discourse.
a,b,c	Individual elements of the Universe of Discourse.
x	System output after defuzzification.
μ(i)	Firing strength of individual rules.
α(i)	Individual rule output.
t	Current temperature.
T	Desired temperature.
dt	Temperature change per one sample period.

Chapter 1

Introduction

1.1 Motivation and Background

Natural language and human thought patterns rely heavily on relative and imprecise terms to describe the world. Statements such as "It is warm outside today" are hard to model in computers and modern control systems. These statements lack precision and can contain terms with ambiguous meanings. Warm means one thing to one person and something completely different to someone else. Is there a boundary between what is hot and what is cold? How about what is hot and what is merely warm? To model a system with parameters such as these, the system must be described in precise and unambiguous terms unless a method exists for interpretation and manipulation of relative terms [1].

Assume an example in which all temperatures above 80°F are considered hot and all temperatures equal to or below 80°F are considered cold. This implies that a temperature of 80.0°F is cold and a temperature of 80.1°F is hot. The difference of 0.1°F would not be perceivable by a human being but would define membership in two entirely different sets. It would be more natural to say that a temperature of 80.0°F is 50% hot and 50% cold. There would be almost no difference between the interpretations of the two temperatures if this were the case.

Fuzzy sets allow partial set membership and allow for ambiguity in boundaries between members and nonmembers of a set. An element can be a partial member of one or more sets. Fuzzy sets infer fuzzy logic, in which operations are defined for logical elements that are partly true and partly false. Fuzzy logic and fuzzy sets infer fuzzy control, which is the design of control systems using fuzzy terms.

A properly defined control system can benefit from the ambiguity permitted by fuzzy rules. A control system based on fuzzy set membership and fuzzy logic can be designed with merely a superficial knowledge of the plant to be controlled. This allows for increased modularity and easier, less math-intensive design of controllers [2,3]. The most challenging part of designing a fuzzy controller is the fuzzification of the inputs and the defuzzification of the outputs. Fuzzification and defuzzification are achieved by defining and perfecting the rules of interpretation and membership functions for the system [1]. For example, consider a model of a shower. The input of the shower system is the current temperature of the water and the change in temperature per unit of time. The output of the system is a control signal for a set of solenoid-controlled valves which determine the mixture of the water and, subsequently, the temperature. This system can be modeled with fuzzy terms if the input is reinterpreted to be hot, warm, cool, cold, or any combination of the previous terms. The current temperature along with the temperature change per unit time can be used to determine a control signal. The fuzzy controller determines if the water should be colder or warmer based on the input and adjusts the output accordingly. The output is then converted to a control signal in a process called defuzzification. The end result is a control signal for each solenoid. The benefit is that the designer needs little information about the system to be controlled and the resulting controller is modular and can be used in other similar systems with only minor modifications.

1.2 History

The notion of fuzzy sets, fuzzy logic, and fuzzy control has been around for about forty years. In 1965 Professor Lotfi Zadeh of the University of Califormia at Berkley wrote a paper titled *Fuzzy Sets* in the journal *Information and Control*. The paper presented and defined the concept of a fuzzy set as well as operations that can be performed on those sets. The first successful application of fuzzy control was not seen until 1975 when a fuzzy-controlled cement kiln came online in Denmark [4,5].

Fuzzy control suffered from a lack of popularity in the United States until recently because of the imprecise nature of the control scheme and a perceived relationship to artificial intelligence, which was not popular in the United States at the time [4]. There are many applications of fuzzy control available today and it is gaining in popularity as it becomes a more credible control method.

It must be noted that there are many areas of future research pertaining to fuzzy control. The questions of stability and optimality have yet to be answered to any degree of certainty.

1.3 Empirical Experiment

In order to present a practical example of the material covered in this work, an empirical experiment is designed and constructed. A model of a confined space in which temperature is to remain constant is constructed. The confined space is created in the form of a box containing one cubic foot of air. The air temperature is increased and decreased with a heating element and cooling system which are housed within the cubic foot enclosure.

The fuzzy controller is designed and implemented on a Nios II microprocessing platform, part of an Altera Cyclone II FPGA. The FPGA and Nios II processor allow for easy reprogramming and are scalable to support larger designs. The schematics and source code for the experiment and the controller design are provided. The results of an experiment designed to show that the fuzzy controller is a viable control scheme are also presented.

1.4 Organization

The first chapter of this paper presents an introduction and motivation for studying fuzzy control techniques as well as an introduction to the system whose design and optimization are detailed in Chapter 5. The second chapter introduces the concept of fuzzy sets and related mathematical functions. The third chapter deals with fuzzy logic and fuzzy-based logical operations. An introduction to fuzzy control schemes is provided in Chapter 4. The fifth chapter presents an empirical example of a fuzzy control system along with experimental results. Appendices provide background information and source code for the example system.

Chapter 2

Fuzzy Sets

2.1 Introduction to Fuzzy Sets

A classical set, as described in Appendix A, is defined as a grouping of elements with similar properties. All possible members define a universe of discourse. Every element of the universe of discourse is either a member of the subset or not a member of the subset. The Venn diagram in Figure 2.1 shows a subset of numbers (A) contained within a universe of discourse. The boundary is crisp and unambiguous and every element is defined as a member or a non-member. Figure 2.2 shows two sets (A and B), both subsets of the universe of discourse. The two sets do not contain any of the same elements. All elements of the universe of discourse are either a member of A, a member of B, or a member of neither. There are no members that belong to both A and B [6,7,8].

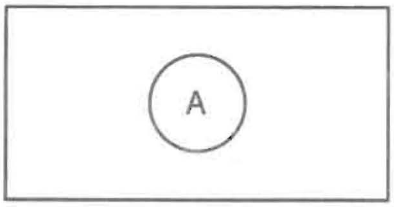

Figure 2.1 Venn Diagram and Single Subset.

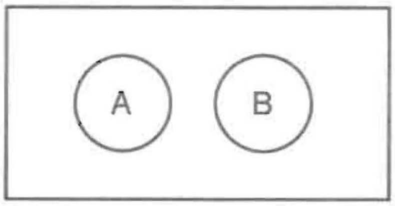

Figure 2.2 Venn Diagram and Two Subsets.

If a subset, C, is defined such that it that contains all of the elements that are members of either subset A or subset B, then C is referred to as the union of subset A and subset B.

$$A \cup B = C \tag{2.1}$$

The shaded region of Figure 2.3 represents the subset C.

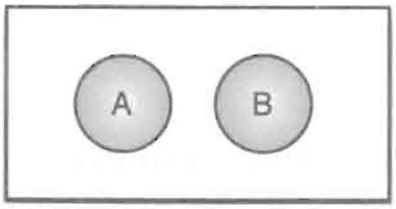

Figure 2.3 Venn Diagram Representing A Union B.

Figure 2.4 represents two sets (A and B) contained within the universe of discourse. The two sets are seen to overlap, representing a small subset of elements which are members of A as well as members of B. This small subset is referred to as the intersection of subset A and subset B.

$$A \cap B = C \tag{2.2}$$

The shaded region of Figure 2.4 represents the intersection of subset A and subset B. Elements in the shaded region are members of subset A and subset B.

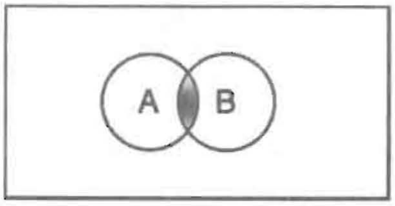

Figure 2.4 Venn Diagram of A Intersection B.

Classical set theory makes a very clear distinction between membership and non membership in a subset (see Appendix A for a detailed introduction to classical set theory). Reality does not follow such crisp boundaries. For example, consider a universe of discourse consisting of all students in a class. Define two subsets {*X*, *Y*} where *X* is the subset of all students considered tall, and *Y* is the subset of all short students [9].

Defining the membership of the sets using classical logic would result in two subsets with no intersection. One subset would contain all people who are considered tall and the other would contain all people who are considered short. There would not be anyone that is considered both tall and short. Figure 2.5 represents this situation [9].

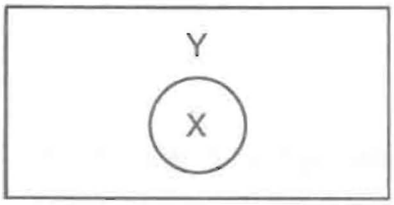

Figure 2.5 Venn Diagram Representation of Tall/Short Example.

How does one classify people who are of exactly average height? Figure 2.6 shows what it would look like if the boundary were made less crisp and some ambiguity was allowed. The gradient shaded region represents a group of people who are partially tall as well as partially short.

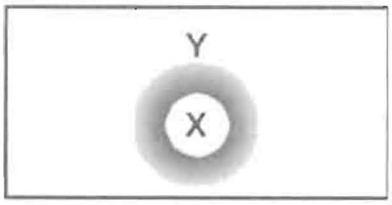

Figure 2.6 Venn Diagram Representing a Fuzzy Set Boundary.

Figure 2.7 shows three possible elements {*a*,*b*,*c*} and their representative position in the universe of discourse. Element *a* is clearly outside the boundary and would therefore be considered short. Element *c* is clearly within the boundary and would therefore be considered tall. Element *b* falls within the shaded border and is therefore not entirely a member of either set. The element is defined as partially contained within *X* and partially contained within Y. The relative percentage of membership is defined with a mathematical function called a membership function.

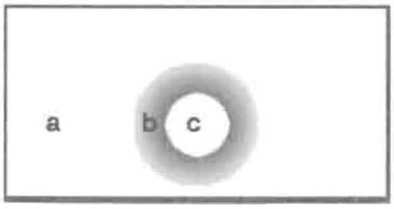

Figure 2.7 Venn Diagram Representation of Three Elements in a Fuzzy Set.

2.2 Membership Functions

Set membership is defined with a mathematical function aptly named a membership function. Elements of a universe of discourse that are being grouped into subsets were traditionally either included or not included in the subset. If it is desired to make them partially included in the subset there must be a definition of the degree of membership. The membership function is used to make the conversion from classical, unambiguous set theory into fuzzy set theory. The creation of the membership function plays a very significant role in the performance of the fuzzy system. Membership functions must be carefully chosen and tuned appropriately to maximize efficiency and performance [1,10,11,12].

Membership functions are almost always represented graphically. The axis of abscissas represents all values in the specified universe of discourse. Generally, this axis should range from the lowest possible value to the highest possible value for any defined universe of discourse. The ordinate axis is a normalized representation of the degree of membership. It may be easier to think of the ordinate axis as a percentage of membership as the values range from 0 to 1, inclusive. The function relates the element value to its respective membership degree.

Figure 2.8 shows what a membership function would look like when applying classical set theory. The function does not allow for partial membership, the set includes all numbers greater than or equal to 0.5. In comparison, Figure 2.9 shows a sample of a trivial fuzzy membership function. The values corresponding to the membership functions in Figure 2.8 and Figure 2.9 are tabulated in Table 2.1 and Table 2.2, respectively.

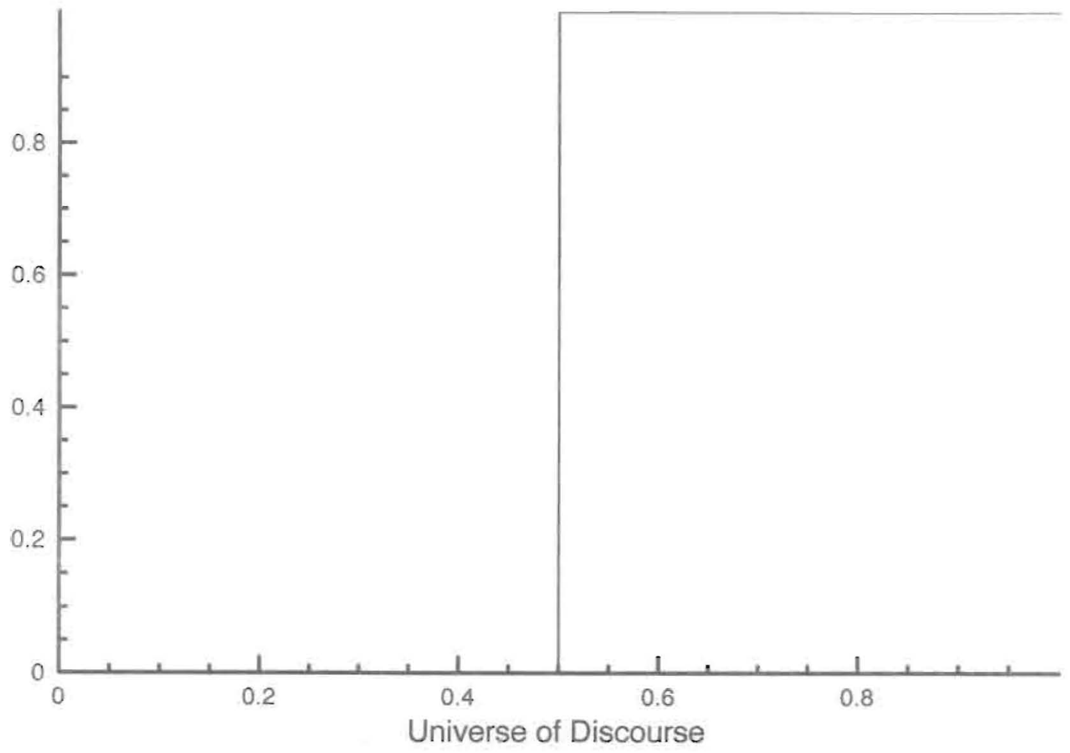

Figure 2.8 Membership Function of Classical Set.

Table 2.1 Membership Values for Figure 2.8.

Element	Membership
0	0
0.499	0
0.500	1

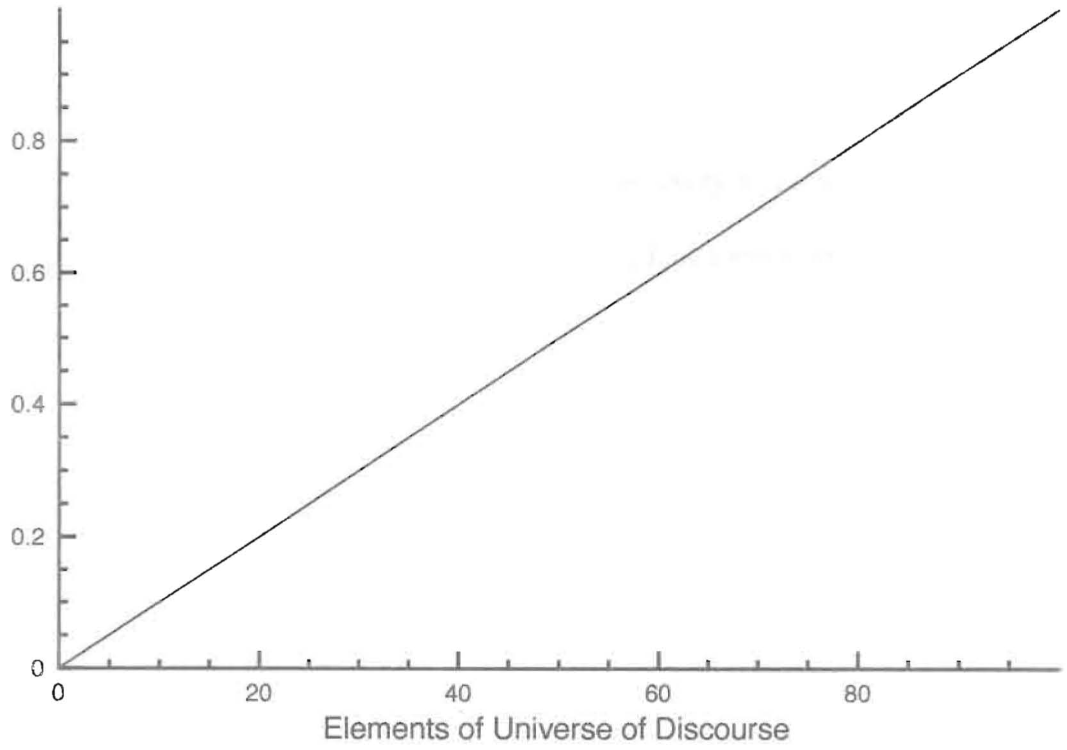

Figure 2.9 Simple Linear Membership Function.

Table 2.2 Membership Values for Figure 2.9.

Element	Membership
0	0
20	0.2
1	1

The example in Figure 2.9 is trivial and only defines membership in one subset. It is possible and usually desired to expand this to multiple subsets. An example of such membership functions is depicted in Figure 2.10 along with associated numerical values listed in Table 2.3. Each line represents a subset and the corresponding membership value for each element in that subset. Since classical rules no longer apply it is common and even desired to have an element be a member of multiple subsets. Consider the membership function in Figure 2.10 representing membership in two subsets {hot, cold}. The universe of discourse is comprised of all possible temperature readings. It can be noted that in this example the total membership of every element is equal to unity but this is merely a coincidence and is in no way required for a membership function.

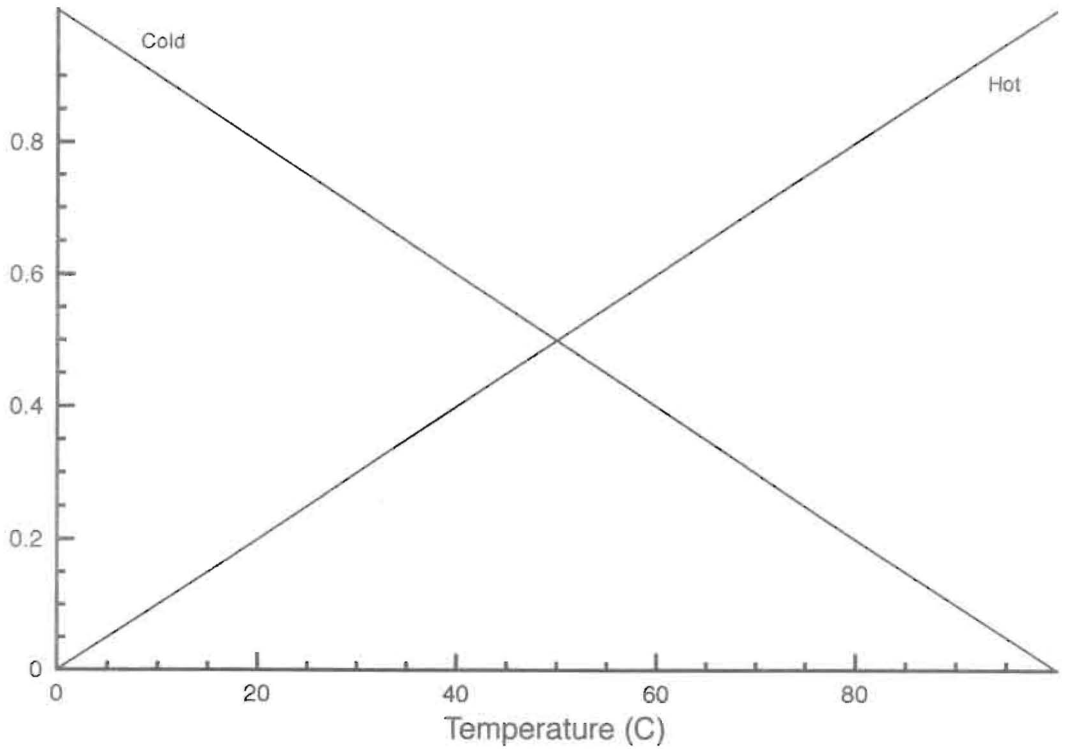

Figure 2.10 Two Set Membership Function.

Table 2.3 Membership Values for Figure 2.10.

Element	Hot Membership	Cold Membership
0	0	1
25	0.25	0.75
75	0.75	0.25
100	1	0

Membership functions can take many forms, depending largely on the system being modeled. The choice of the membership function is made by the engineer designing the system and has many consequences on the performance of the system. The consequences of such decisions are elaborated upon when discussing input fuzzification in Chapter 4. Some common forms of membership functions are shown in Figures 2.11-2.13. This list of functions is by no means inclusive and functions can be created for virtually any system [9]. Also, it is often desirable to combine membership functions depending on the system being modeled.

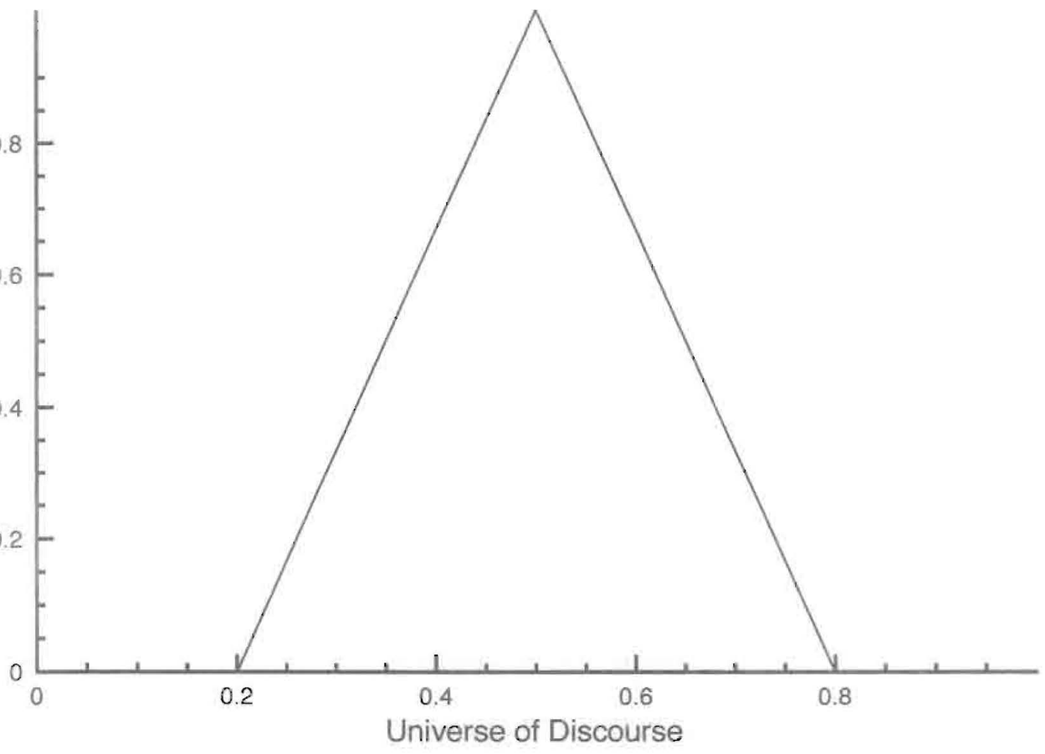

Figure 2.11 Triangular Membership Function.

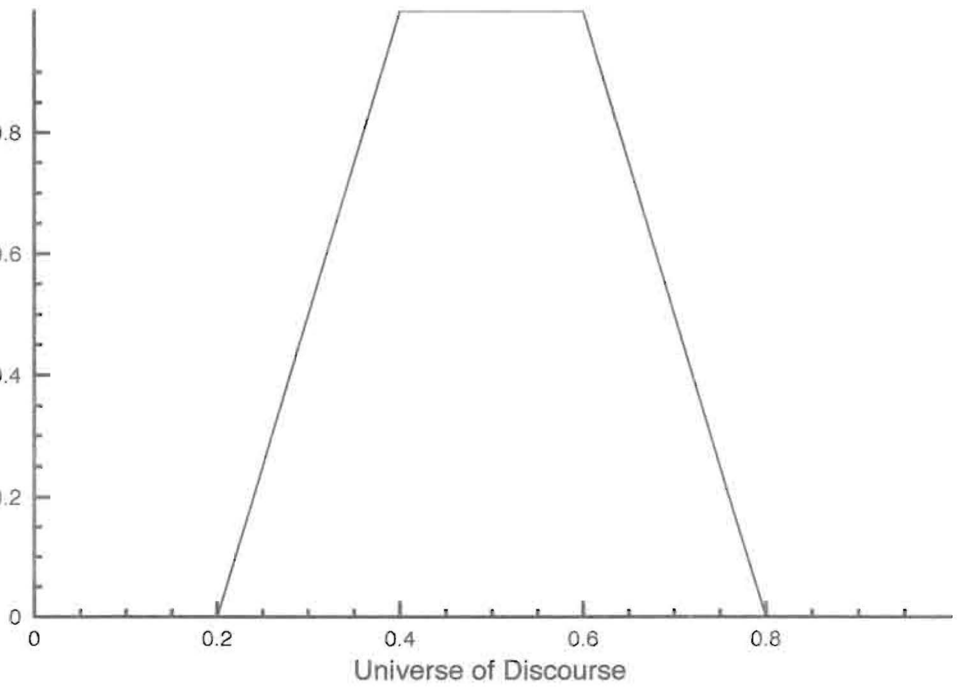

Figure 2.12 Trapezoidal Membership Function.

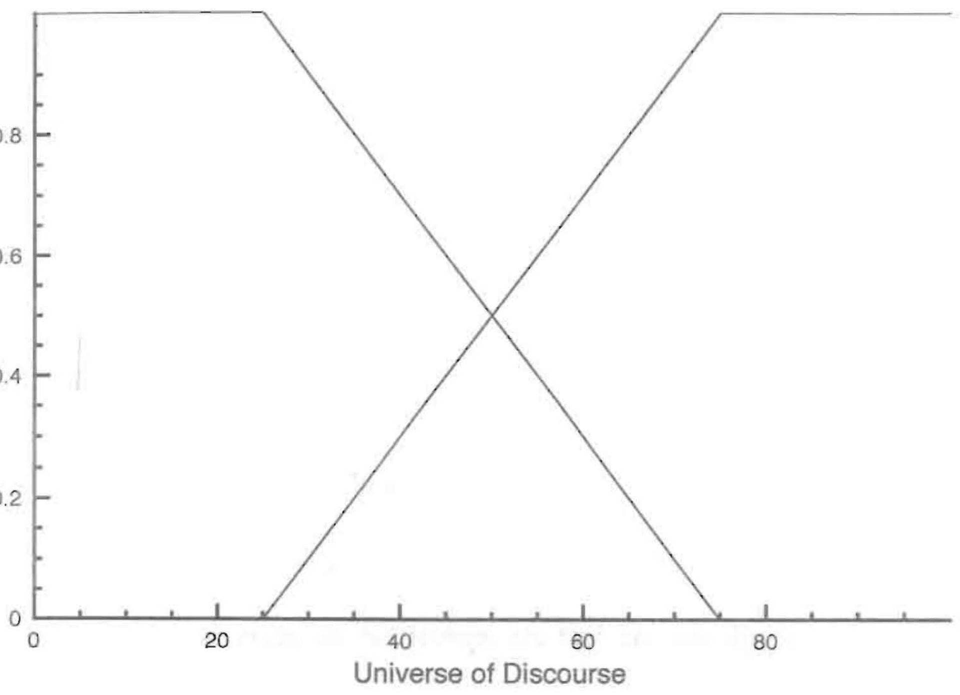

Figure 2.13 Two Trapezoidal Membership Functions.

Chapter 3

Fuzzy Logic

3.1 Introduction to Fuzzy Logic

Classical logic is concerned with deriving truth or falsity from propositions and combinations of propositions. The notation and operations performed on classical logic statements are described in more detail in Appendix B. A hallmark of classical logic is that every statement is either true or false. There can be no statements which are partly true or partly false, and certainly no statements that are equally true and false. This limitation is addressed with the introduction of fuzzy logic [1].

A parallel can be drawn between the relationship between classical and fuzzy sets and the relationship between classical and fuzzy logic. Consider a subset of the universe of discourse which consists of all elements which can be proven to be true. In classical set theory, each element is either a member or not a member of the truth subset. Changing the rules from classical to fuzzy subsets allows partial membership in the proven true subset as well as the not proven true subset. An element can now be either a full member of one subset or a partial member of both subsets. It is important to note that fuzzy logic fully complies with the rules of classical logic in both extreme cases, no membership and full membership (0 and 1).

The operations of fuzzy logic are described in the following section and the material is borrowed heavily from [9].

3.2 Fuzzy Logic Operations

It was previously noted that in the completely true and completely false cases fuzzy logic follows the rules of classical logic. Consider the truth table in Table 3.1, which represents the truth table for a standard logical AND function. As expected, the statement A AND B is only true if A and B are both true. This classical logic conclusion is preserved if the function is reconsidered to be the minimum of A and B as in Table 3.2. In the extreme case where A and B are both completely true, the result is still completely true. However, the function now allows for partial membership and is no longer compliant with the rules of classical logic [1].

Table 3.1 Classical AND Function.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.2 Fuzzy AND Function.

A	B	A AND B
a	b	min(a,b)

In a similar fashion the OR and NOT functions can be reconsidered as shown in Tables 3.3 and 3.4. The logical OR function is changed to a maximum function, which follows the rules of classical as well as fuzzy logic. The logical NOT function is reconsidered as an algebraic (1 - A) function which also adheres to the classical as well as fuzzy rules.

Table 3.3 Classical OR Function.

A	B	A OR B	max(A,B)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Table 3.4 Fuzzy OR Function.

A	NOT A	(1 - A)
0	1	1
1	0	0

The conversion of the three basic logical operators leads to a discussion on classical and fuzzy logical statements. A statement such as

$$\text{if X is A then Y is B} \tag{3.1}$$

can be interpreted as a classical implication if A and B are subsets in the classical sense. However, if A and B are fuzzy subsets then the statement takes on a new meaning. If membership in A and B were defined using fuzzy subset theory then the elements would only have a degree of membership and the implication above now relates the degrees of membership. Once again, it is important to note that in the extreme case, the fuzzy rules apply to classical logic as well. For example, consider the statement above (3.1). If X has a degree of membership of 0.7 in A then the implication implies that Y has a 0.7

degree of membership in B. In other words, partial antecedents imply partial consequents [6].

The antecedent can contain multiple terms, in which case they must be resolved to a single degree of membership using the techniques discussed earlier in this section. Consider the following example.

Assume that the rules of a given system are:

$$\text{if } x \text{ is } A \text{ AND } y \text{ is } B \text{ then } z \text{ is } C \tag{3.2}$$

Also consider the membership function defined by Figure 3.1. The function defines membership in two subsets $\{A,B\}$. Table 3.5 lists the sampled values and their corresponding membership values. Since x is a 0.7 member of A and y is a 1.0 member of B then z is minimum(0.7 , 1.0) = 0.7 member of C.

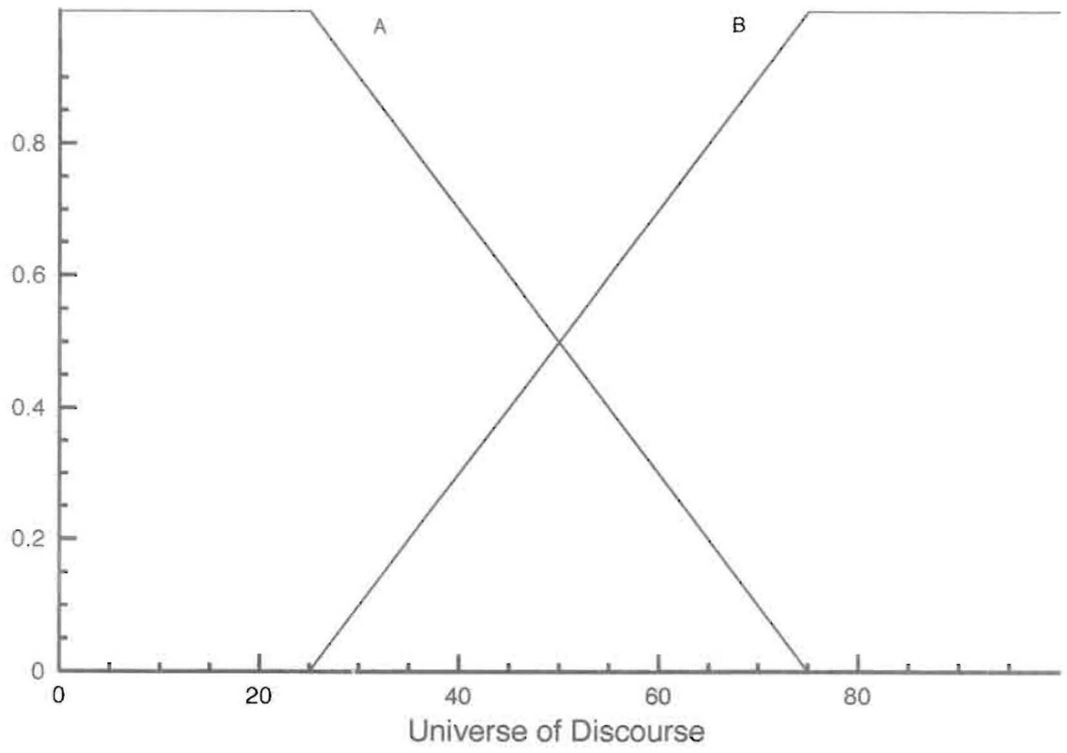

Figure 3.1 Membership Function Defining Membership in Two Subsets.

Table 3.5 Membership Values Derived From Figure 3.1.

	A	B
x = 40	0.7	0.3
y = 80	0	1

Chapter 4

Fuzzy Control

4.1 Introduction to Fuzzy Control

It was previously mentioned that fuzzy control systems can be designed with only a superficial knowledge of the plant to be controlled. Fuzzy control is ideal when a precise mathematical model of the plant does not exist or if the control process using traditional methods is too expensive in terms of required computational power. In these cases, it is often desirable to design a simple system based on rules governed by empirical data and observation [4].

There are six assumptions that an engineer must be willing to make before selecting a fuzzy controller based approach. The following six rules were inspired by Ross [1].

1. The plant is observable and controllable. All state, input, and output variables are available for measurement or computation.
2. There is comprehensive operational knowledge of the plant to be controlled. A mathematical model is not required but knowledge of operation, consisting of operator knowledge and engineering common sense is required to devise a set of literary rules to define operation of the controller.
3. The engineer assumes that a solution exists.
4. The fuzzy controller is be "good enough" to accomplish the desired result but may not be the optimal solution.

5. The precision of the controller is limited and must be designed within an acceptable tolerance range. One objective of the fuzzy controller is to minimize computation and computing cost so an overly-precise system will negate this advantage.
6. There is no accepted methods of determining stability or optimality for fuzzy controllers. This is an area for further study. The designing engineer should thoroughly test the system for instability and inaccuracy before finalizing the design.

Once the decision to use a fuzzy controller is made there are several steps to follow in order to design and implement the controller. The steps, presented below, are borrowed from [1] and are the topic of discussion for the rest of this chapter.

1. Identify all system variables. Inputs, outputs, and all state variables must be identified and defined.
2. Identify and partition the universe of discourse into smaller, fuzzy subsets. Assign each subset a linguistic title for later use.
3. Create a membership function for each fuzzy subset.
4. Create the rule base for the system by determining the relationship between the input fuzzy subsets and the output fuzzy subset.
5. Normalize the inputs to fall within the tolerance interval defined for the system.
6. Fuzzify the inputs.
7. Implement fuzzy logic to determine the output for each rule in the rule base.
8. Combine the output of all rules based upon the aggregation method chosen for the system.
9. Defuzzify the outputs to produce a crisp output signal.

The procedure provided above works well for simple fuzzy systems and is discussed in greater detail in what follows. The design example presented in the next chapter also follows this design procedure.

4.2 Preliminary Design

The design procedure from the previous section indicates that the first step in the design process is to identify variables and states. This step may seem trivial but it is important to consider the hardware that one will implement and choose sensors and other inputs accordingly. If the system does not have enough sensors it can not make accurate control decisions. Too many sensors will dramatically increase the complexity of the system and also the computational effort required to control the system. Therefore, balance must be struck and the optimal set of inputs and states must be defined.

As an example, consider the system presented in the next chapter. The system implements a fuzzy controller for a temperature-control process. At first glance, it may seem necessary to know the ambient temperature of the air outside of the box, however that would add a sensor input. This sensor input must be fuzzified and also contained in the rule base. It is much more reasonable to only measure the temperature of the confined space and also extract, from that data, the temperature change per unit time. Those two inputs would only require one sensor and would minimize the amount of computations required for the system to be controlled.

Choice of sensor is very important. Since the system will fuzzify the inputs it does not make good design sense to have only binary sensors as inputs because it would negate the primary advantage of the fuzzy controller; the fuzzy controller takes an analog input and uses the rules of the system to develop an analog output. Although the binary case is still able to be handled with a fuzzy controller it may be easier and less expensive to implement with combinational logic [1].

The second step of the design procedure consists of identifying and partitioning the universe of discourse. As discussed in Chapter 2, the universe of discourse consists of all possible input values for the system being designed. Careful consideration should be given to ensure that all possible values are accounted for. All values in the universe of discourse should be at least a partial member of one of the subsets to avoid the possibility of the controller producing an indeterminate output signal. The choice of subset title is up to the designing engineer but should be descriptive as it will be used to create the membership functions and the rule base that defines the system.

Each fuzzy subset will generate one curve on the membership function. The choice of the shape of the curve is to be made by the design engineer based on knowledge of how the system operates. The membership function curve for each fuzzy subset will be tuned to produce the most accurate output signal. The triangular shape is the most commonly used and can be properly tuned for most systems but there are other possibilities as discussed previously [9].

The third step of the process consists of creating membership functions for each input to the system. Section 2.2 provides background information on the creation of membership functions. The creation of the membership function is more of an art than a science and engineers and operators with more experience in how the system operates would be best suited for this task because the design engineer must rely on experience and a limited knowledge of the plant to optimize the membership functions [2]. The design example in the next chapter details the creation and tuning of the membership functions used to define the system.

4.3 Rule Base Creation

In what follows, the creation of a rule base, briefly discussed in Section 3.2, is now presented in more detail. The rule base of a system is a set of linguistic logical rules that define the system. The rules are presented as linguistic statements and take the form of *if-then* statements. The rules of the system essentially define the knowledge of the operator. For example, the operator of a machine will know what to do to compensate for variations in output. The knowledge of the operator is based on past experience and common sense. Defining the rule base of a system requires that the designer have a thorough knowledge and, preferably, experience controlling the process manually. Thus, if a controller is being designed for an existing process, the design engineer should consider consulting with the operator of the machine or process to utilize their expert knowledge and experience[13].

The number of rules varies based on the complexity of the system being considered. A more complex plant with more sensor inputs requires more rules to properly define the controller. It is a good design practice to derive rules for all possible input states [14]. As an example, consider the following rule-set for a system which consists of a temperature measurement from a water-mixing chamber. The membership function for the temperature measurement defines membership in three fuzzy subsets: hot, neutral, and cold. The change in temperature is also considered in the rules for the system. The output is a control signal for two pumps, one pumping hot water and one pumping cold water. All possible scenarios for the system are considered and the rule base is complete as follows:

- *If water is hot and getting hotter, then output is very cold (VC).*
- *If water is hot and getting colder, then output is cold (C).*
- *If water is hot and not changing, then output is cold (C).*
- *If water is neutral and getting hotter, then output is cold (C).*
- *If water is neutral and not changing, then output is not changing (NC).*
- *If water is neutral and getting colder, output is hot (H).*
- *If water is cold and getting hotter, then output is hot (H).*
- *If water is cold and not changing, then output is hot (H).*
- *If water is cold and getting colder, then output is very hot (VH).*

The preceding system of rules could be presented in tabular format to make them easier to follow and easier to construct. Consider Table 4.1, the same data is presented in a much more compact format which is easier to follow. Each row of the table represents the water temperature and each column represents the temperature change. The value in each cell of Table 4.1 indicates the output control signal.

Table 4.1 Water Temperature Change Rule Matrix.

	Colder	Not Changing	Warmer
Hot	C	C	VC
Neutral	H	NC	C
Cold	VH	H	H

Defining the rule base properly is essential for accurate operation of the controller. The rule base will be used to determine the system output through defuzzification. The output of the rule base is processed to form the output signal through the process of defuzzification, which is discussed in more detail in Section 4.5

4.4 Input Processing

At this point in the design process, it is essential to consider the electrical signals that will be forming the control system. The maximum ranges of all signals should be considered to ensure compatibility with the hardware being used to implement the system. Step five of the previously detailed design process states that all inputs must be normalized to the system. The process of normalization is crucial, particularly where analog to digital conversions are being conducted. An analog to digital converter has a specific range of input values that it can convert to digital signals. If the range of the signal being supplied to the analog to digital converter is outside of the acceptable range of the converter then the digital output will not accurately represent the signal and thus the controller accuracy will be compromised. If the range of the input signal is much narrower than the range of the analog to digital converter, then the output will not be as accurate as possible and the analog to digital converter is not being used to maximum efficiency.

4.5 Logic Implementation and Rule Inference

Once all input signals are normalized and converted to their digital approximation through the use of the appropriate analog to digital converters, the next step is to begin designing the code and algorithms that define the system. Of course, the syntax of the code will differ based on the hardware that is being used to implement the system, thus will be omitted here. However, the code syntax for the experimental system, implemented as part of this work, is fully discussed in Chapter 5 and Appendix C.

The block diagram in Figure 4.1 presents a conceptual block diagram of a simple fuzzy controller. The controller has one analog input and also uses the change in the error signal as an input. There are three rules defining the system and one analog output.

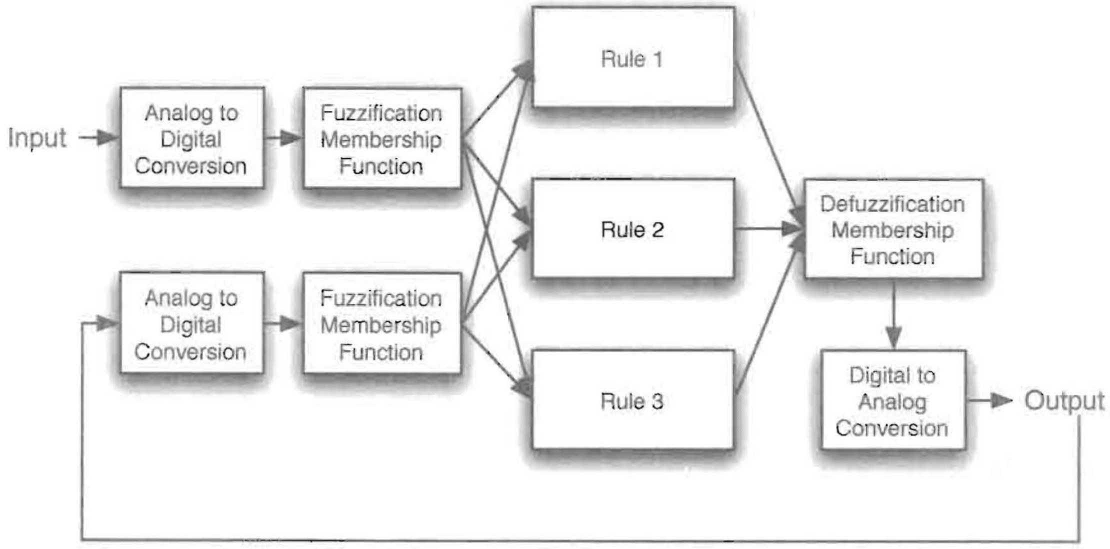
Input
Analog to Digital Conversion
Fuzzification Membership Function
Analog to Digital Conversion
Fuzzification Membership Function
Rule 1
Rule 2
Rule 3
Defuzzification Membership Function
Digital to Analog Conversion
Output

Figure 4.1 Sample Fuzzy Controller Block Diagram.

Each cell in the rule matrix in Table 4.1 corresponds to a rule. Each rule will have a truth value based on a chosen method of inference. A common method is the MIN-MAX method in which the output is either calculated with a minimum or maximum function based on the construction of the defining rules. If the rules list two antecedents which are combined with an AND operator then the minimum value is used. Consequently, if the OR operator is used, the output is determined with the use of a maximum function. In the rule matrix presented previously, the AND operator was used, so each cell in Table 4.1 will have a corresponding numerical value assigned to it [14]. For example, consider the first rule:

- *If water is hot and getting hotter, then output is very cold.*

Hot and *getting hotter* are the names of two fuzzy subsets. Each of these has a corresponding membership value, which is passed into the rule base from the input fuzzification membership functions. Assume the the input fuzzification membership functions describe membership in the *hot* subset at a value of 0.8 and membership in the *getting hotter* subset at a value of 0.3. Since the rule states that both values must be true, through the use of a logical AND, the minimum function should be used. The minimum (0.3,0.8) function yields a result of 0.3 so the called "firing strength" of the rule is 0.3

The example rule base in Table 4.1 has a total of nine rules which must be evaluated. Each rule will produce a firing strength. The rules must be combined to produce one output signal for each of the pumps in the example system. The combination of the firing strength values is achieved by the use of output inference and defuzzification.

One popular method of defuzzification of the output value is called the centroid method. The centroid method is not the only method for defuzzification of output but it is a useful and common one so it is the only method discussed and used in this work.

The general equation defining this method is shown in Equation 4.1. The output is denoted as x.

$$x = \frac{\sum_{i=1}^{n} \mu(i) * \alpha(i)}{\sum_{i=1}^{n} \mu(i)} \tag{4.1}$$

where μ(i) is the firing strength and α(i) is the output value of the rule.

The output value associated with each rule is determined from the output membership function in an inverse fashion to the way in which the input membership values are determined. Input membership values are determined by finding the input value on the axis of abscissas and mapping them, through the membership function curves, to the membership value on the ordinate axis. Inversely, the output value for a rule is determined by the value on the axis of abscissas in which the corresponding ordinate value is at its maximum. Obviously, this means that the output membership functions must also be fine-tuned for maximum efficiency of the system [4].

4.6 Output Processing

At this final point in the initial design stage the only design decisions left to be made pertain to the output signals and interfacing them to the equipment to be designed. The signals may need to be converted to analog signals through the use of digital to analog converters or may be used in their digital form. The design at this stage is left to the design engineer and requirements for the system to be designed.

At various points in the preceding discussion it was mentioned that several system components should be fine-tuned after the initial design is complete. These components include the input fuzzification and output defuzzification membership functions as well as the rule matrix. It should also be noted that the topic of stability has not been discussed because there is no definitive way of determining fuzzy system stability. The designing engineer should fully test the system to determine that the system is stable and operates within the design parameters[15,16]. Testing the system under all possible conditions is necessary to verify that the design is operating within design parameters and to ensure that there is no undesired operation. Although this may be a lengthy process for some controllers, it is the only method available for verifying stability.

Chapter 5

Design and Implementation

5.1 Motivation and Introduction

In order to provide an empirical example and to show that fuzzy control is a viable control scheme a fuzzy control system was designed and built. The fuzzy control system software was designed using the procedure described in Chapter 4 and is presented in this fashion throughout the remainder of this chapter.

The hardware was designed for the NIOS II soft processing core on an Altera Cyclone II FPGA. This system was chosen to stay true to the modular and versatile nature of the fuzzy controller. The processing core as well as all control logic can be changed by simply reprogramming the FPGA. The hardware design procedure and all supporting source code and documentation is provided at the end of this chapter.

The plant being controlled was a model of a heating and cooling process to maintain a constant air temperature in a confined space. In order to simplify the design process the confined space in which the temperature was controlled was a small box with one cubic foot of air inside the box. The ambient temperature was increased with the use of a modified hair dryer blowing heated air into the enclosure and lowered by one DC fan blowing room temperature air into the enclosure and one DC fan exhausting the air inside of the enclosure.

5.2 Controller Design, Step 1

The first step of the design process, as presented in Chapter 4, was the identification of input/output variables and states. The controller had five output states. The states corresponded to all useful combinations of output heating and cooling. The five states are were follows:

- High Heat (HH)
- Low Heat (LH)
- All Off (AO)
- Low Cool (LC)
- High Cool (HC)

The High Heat state was achieved by activating the hair dryer on the high setting and the Low Heat state was achieved by activating the hair dryer on the low setting. The Low Cool state was one DC fan blowing air into the confined space and the High Cool state consisted of two DC fans, one moving air into the space and one removing air from the space. The All Off state merely consisted of all devices turned off.

The two variables in the process were defined to be current temperature, *t*, and temperature change, *dt*. The temperature was measured directly with a temperature sensor placed within the confined space. The temperature change is calculated as the change in temperature over one sampling period.

5.3 Controller Design, Step 2

The second step of the design procedure was to identify and partition the universe of discourse into fuzzy subsets.

The universe of discourse consisted of all possibly measured temperatures and all possible temperature changes. The universe of discourse was partitioned into fuzzy subsets as defined below. The elements of the subsets were variable because the desired temperature, *T*, was defined as a variable in order to maintain the versatility of the controller. The fuzzy subsets chosen for this experiment were:

- Hot
- Warm
- Neutral
- Cool
- Cold
- Heating
- Constant
- Cooling

5.4 Controller Design, Step 3

The third step of the design process was to create membership functions for each of the fuzzy subsets defined in the previous section. The membership functions are all presented individually in Figures 5.1-5.5 and Figures 5.7-5.9. The membership functions are presented together in Figure 5.6 and Figure 5.10. The desired temperature was variable but the membership functions in Figures 5.1-5.10 assume a desired temperature of 60°F. Varying the desired temperature does not change the membership functions, only their axis intercepts.

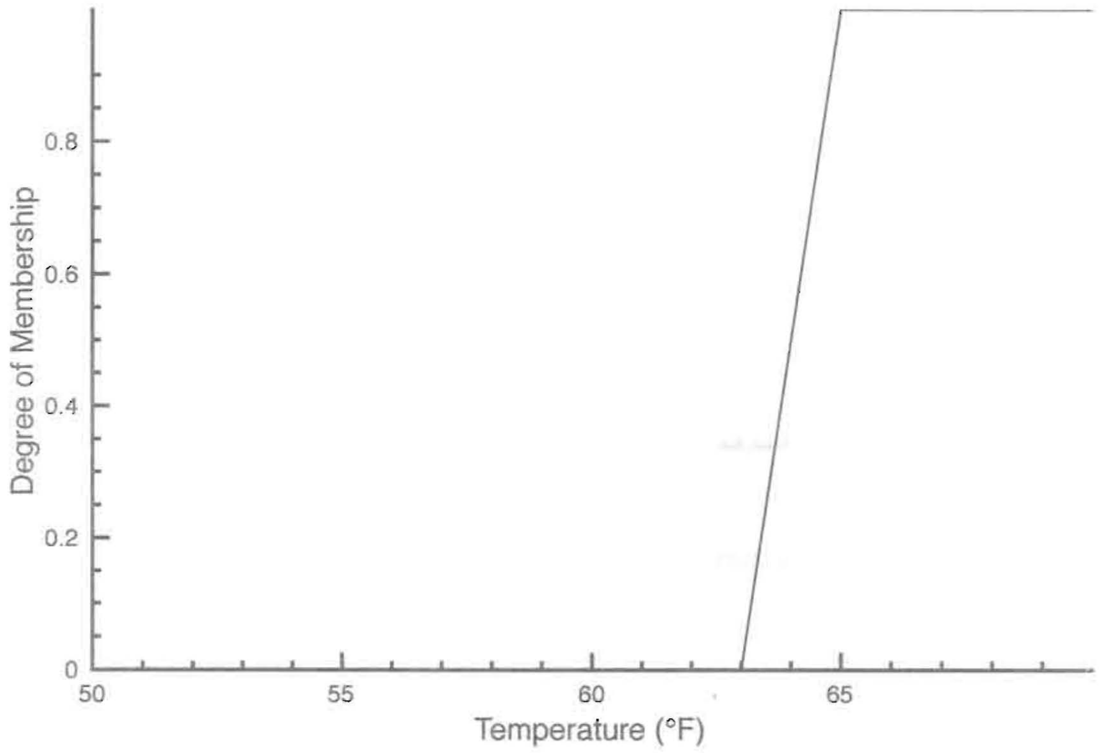

Figure 5.1 Hot Membership Function.

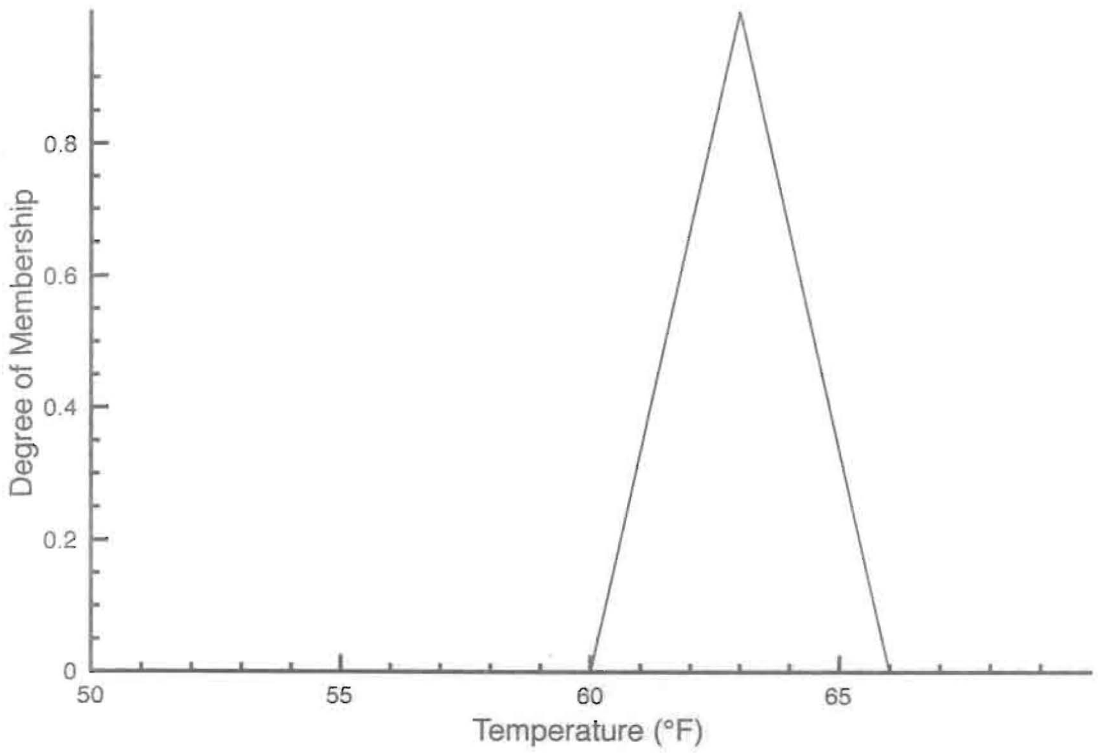

Figure 5.2 Warm Membership Function.

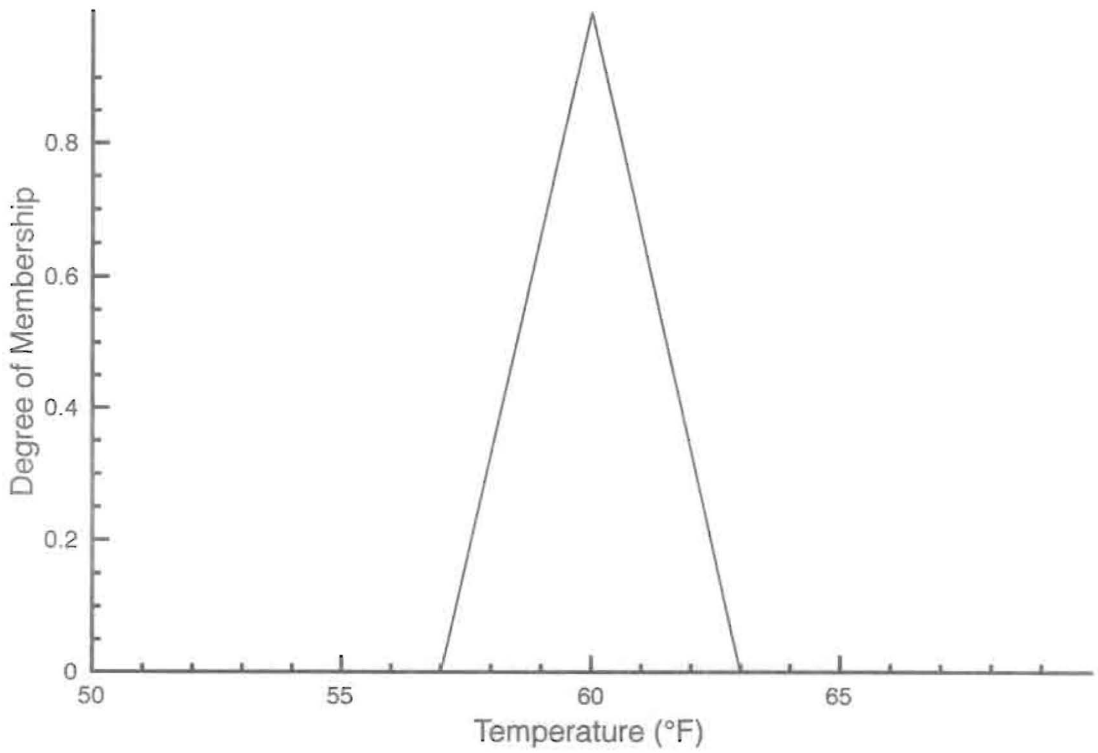

Figure 5.3 Neutral Membership Function.

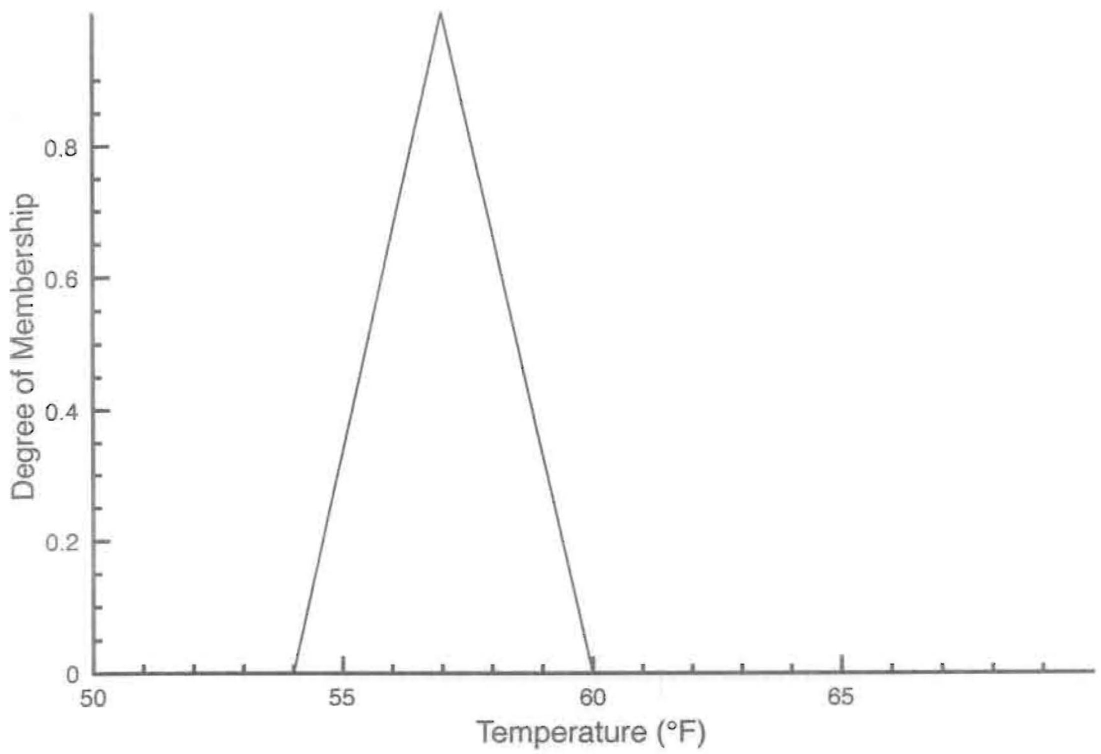

Figure 5.4 Cool Membership Function.

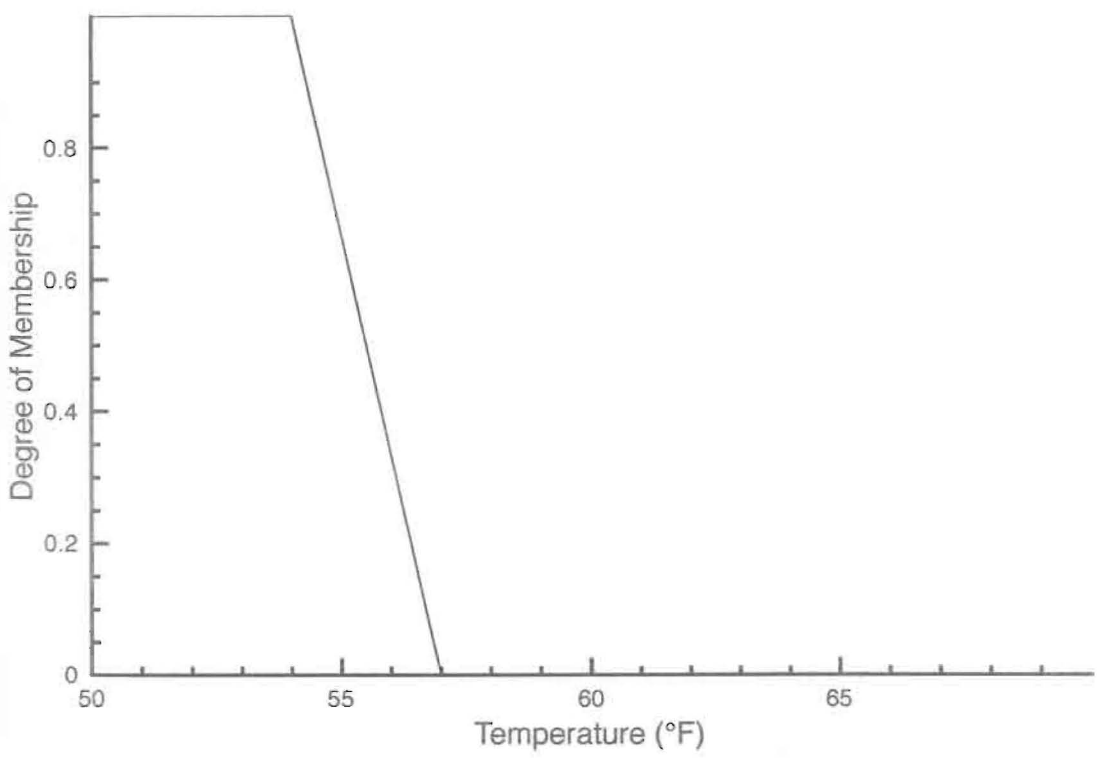

Figure 5.5 Cold Membership Function.

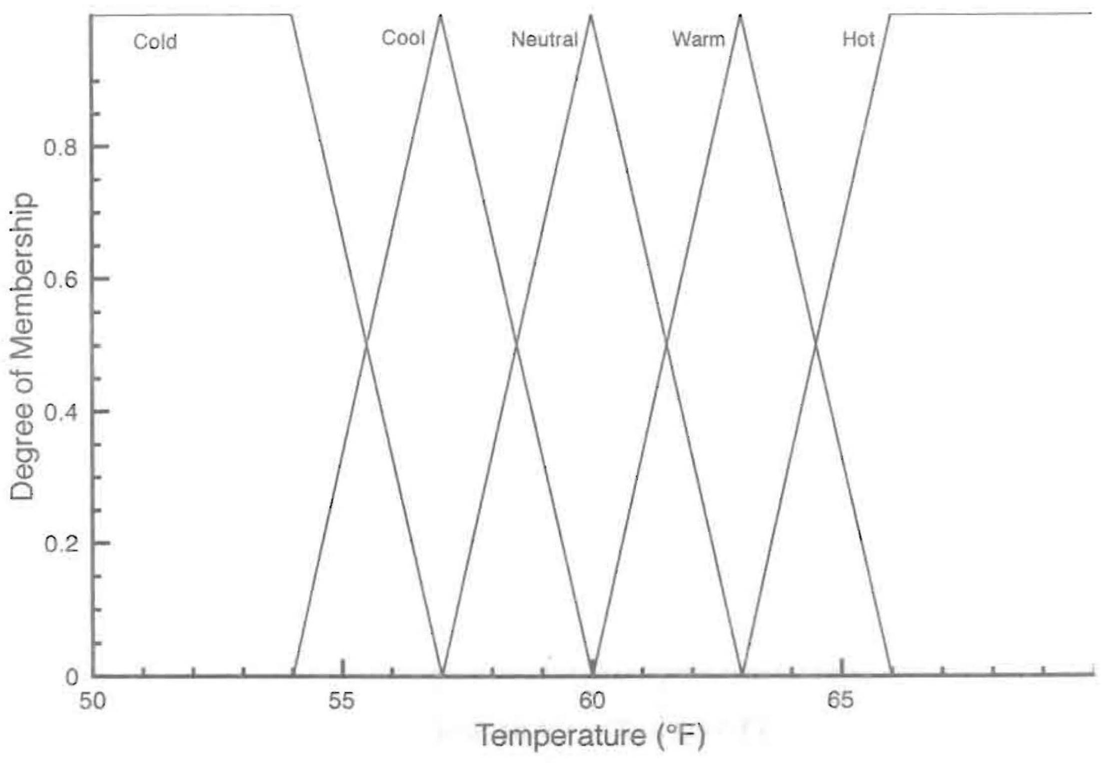

Figure 5.6 Temperature Membership Functions.

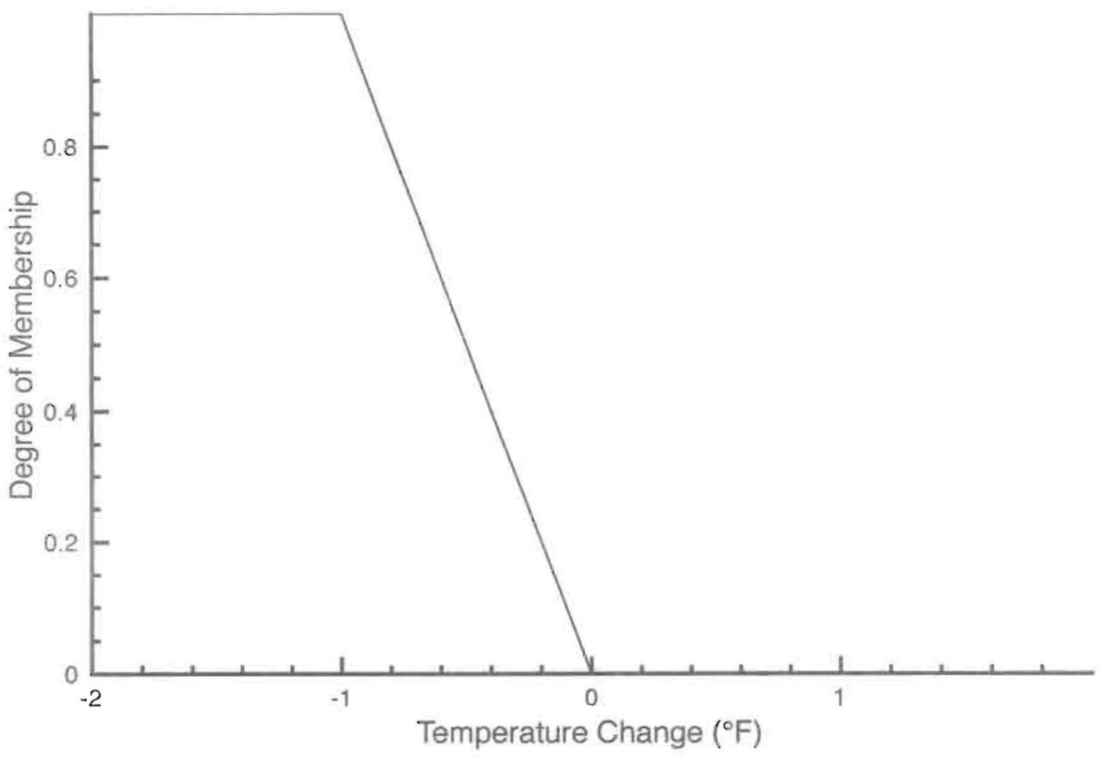

Figure 5.7 Cooling Membership Function.

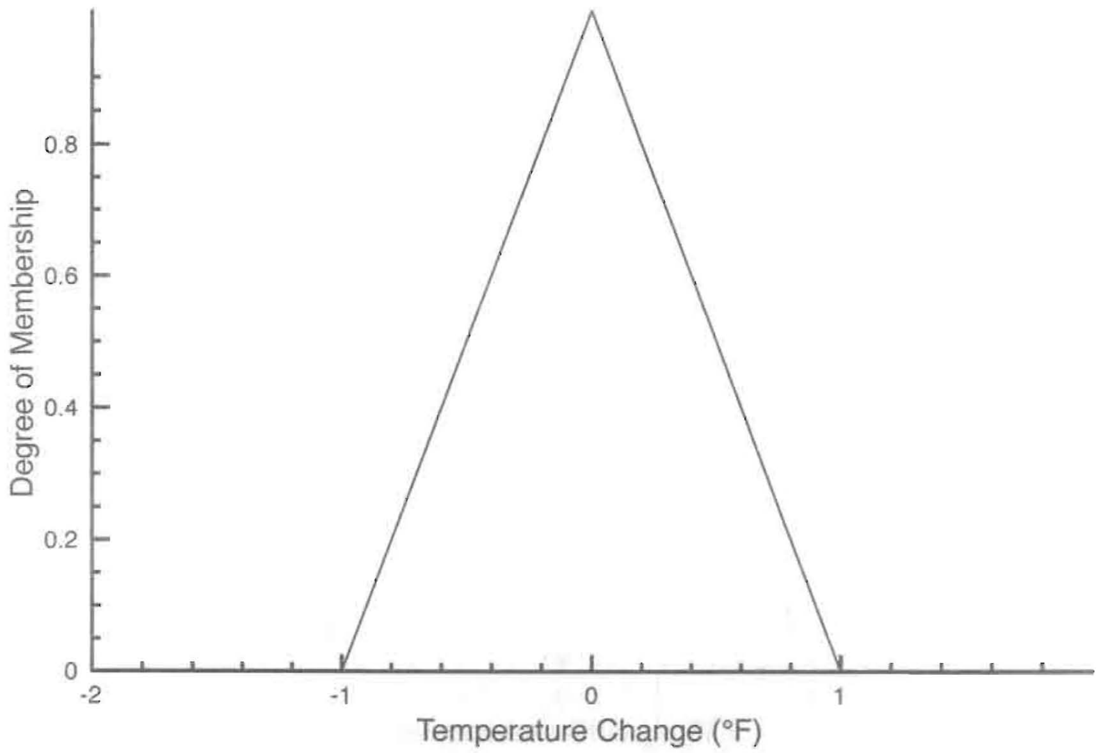

Figure 5.8 Constant Temperature Membership Function.

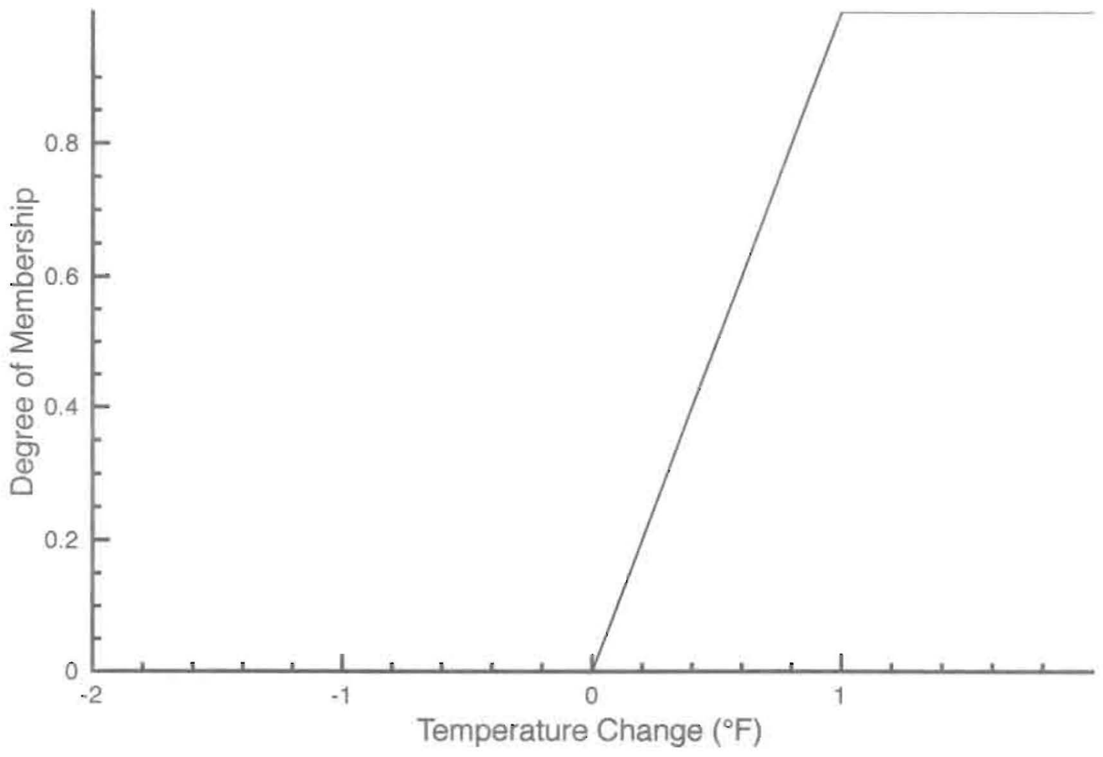

Figure 5.9 Warming Membership Function.

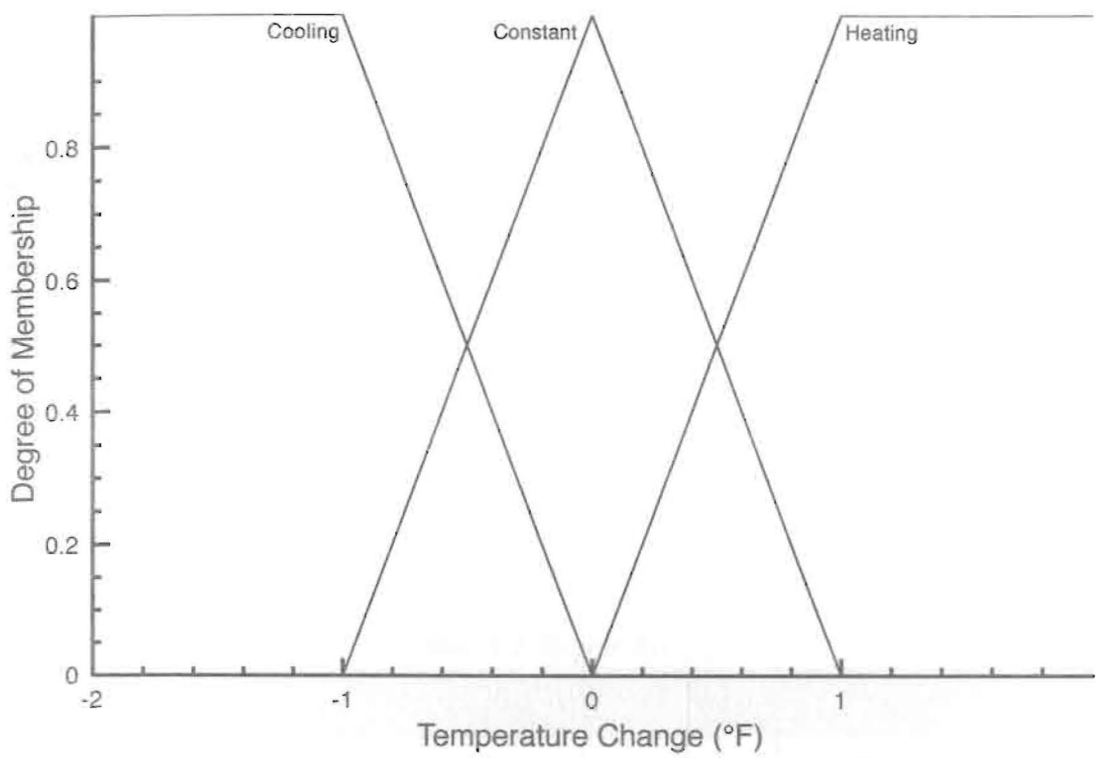

Figure 5.10 Temperature Change Membership Functions.

5.5 Controller Design, Step 4

Step four calls for the creation of the rule base for the system. The rule base for the system was as follows (note that *t* represents current temperature and *dt* represents the change in current temperature over one sampling period):

- If *t* is hot and *dt* is heating, then output is high cool.
- If *t* is warm and *dt* is heating, then output is high cool.
- If *t* is neutral and *dt* is heating, then output is all off.
- If *t* is cool and *dt* is heating, then output is low heat.
- If *t* is cold and *dt* is heating, then output is high heat.
- If *t* is hot and *dt* is constant, then output is high cool.
- If *t* is warm and *dt* is constant, then output is low cool.
- If *t* is neutral and *dt* is constant, then output is all off.

- If *t* is cool and *dt* is constant, then output is low heat.
- If *t* is cold and *dt* is constant, then output is high heat.
- If *t* is hot and *dt* is cooling, then output is high cool.
- If *t* is warm and *dt* is cooling, then output is low cool.
- If *t* is neutral and *dt* is cooling, then output is all off.
- If *t* is cool and *dt* is cooling, then output is high heat.
- If *t* is cold and *dt* is cooling, then output is high heat.

The rule base presented in tabular format is as follows:

Table 5.1 System Rule Base

	Hot	Warm	Neutral	Cool	Cold
Cooling	HC	LC	AO	HH	HH
Constant	HC	LC	AO	LH	HH
Heating	HC	HC	AO	LH	HH

5.6 Controller Design, Step 6

The fuzzification of the input signals was achieved through the use of software. The temperature input signal was read into the processor via an analog-to-digital converter and was then converted to degrees fahrenheit. The temperature change was calculated by subtracting the current temperature reading from the previous one. These two input signals were fuzzified with the use of mathematical functions describing the input membership functions. Those equations are presented in detail in the rest of this section.

The expressions describing each of the membership functions are as follows:

- Hot

$$\mu(t)=\begin{cases}1 & \text{if} \quad t \geq (T+6) \\ \dfrac{(-t+T+6)}{3} & \text{if} \quad (T+6) > t > (T+3) \\ 0 & \text{if} \quad t \leq (T+3)\end{cases} . \tag{5.1}$$

• Warm

$$\mu(t)=\begin{cases}1 & \text{if} \quad t = T+3 \\ \dfrac{t-T}{3} & \text{if} \quad (T+3) > t > T \\ \dfrac{(-t+T+6)}{3} & \text{if} \quad (T+6) > t > (T+3) \\ 0 & \text{if} \quad \mathrm{t} \geq (T+6) \\ 0 & \text{if} \quad t \leq T\end{cases} . \tag{5.2}$$

• Neutral

$$\mu(t)=\begin{cases}1 & \text{if} \quad t = T \\ \dfrac{t-T+3}{3} & \text{if} \quad (T-3) < t < T \\ \dfrac{-t+T+3}{3} & \text{if} \quad T < t < (T+3) \\ 0 & \text{if} \quad t \leq (T-3) \\ 0 & \text{if} \quad t \geq (T+3)\end{cases} . \tag{5.3}$$

- Cool

$$\mu(t) = \begin{cases} 1 & \text{if } t = (T-3) \\ \dfrac{t-T+6}{3} & \text{if } (T-6) < t < (T-3) \\ \dfrac{-t+T}{3} & \text{if } (T-3) < t < T \\ 0 & \text{if } t \leq (T-6) \\ 0 & \text{if } t \geq T \end{cases} . \quad (5.4)$$

- Cold

$$\mu(t) = \begin{cases} 1 & \text{if } t \leq (T-6) \\ \dfrac{-t+T-3}{3} & \text{if } (T-6) < t < (T-3) \\ 0 & \text{if } t \geq (T-3) \end{cases} . \quad (5.5)$$

- Warming

$$\mu(t) = \begin{cases} 0 & \text{if } dt \leq 0 \\ dt & \text{if } 0 < dt < 1 \\ 1 & \text{if } dt \geq 1 \end{cases} . \quad (5.6)$$

- Constant

$$\mu(t) = \begin{cases} 0 & \text{if } dt \leq -1 \\ 0 & \text{if } dt \geq 1 \\ dt+1 & \text{if } 0 > dt > -1 \\ 1-dt & \text{if } 1 > dt > 0 \\ 1 & \text{if } dt = 0 \end{cases} . \quad (5.7)$$

- Cooling

$$\mu(t) = \begin{cases} 0 & \text{if } dt \geq 0 \\ -dt & \text{if } 0 > dt > -1 \\ 1 & \text{if } \mathrm{dt} \leq -1 \end{cases} . \quad (5.8)$$

5.7 Controller Design, Step 7

At this point in the design process it was necessary to implement the logic necessary to determine the output from each rule. Inspection of the rules for the system reveals that each rule was based on a logical AND function. The corresponding fuzzy operator was thus the minimum function. The algorithm calculated the minimum membership value for each antecedent in the rule. The minimum function was repeated for each rule and the result was stored in an array.

5.8 Controller Design, Step 8

The array of rule values created in the previous section was then evaluated to determine the final output from the rule base. The aggregation method used for this process was the maximum function. The maximum function merely took the rule with the highest firing strength and applies the consequent of the statement as the output of the system. In this fashion, step nine was also completed as the defuzzification is inherent in this step.

The controller design process was complete after this step. The system took an analog input in the form of a temperature measurement and converted it to a digital signal that the system can use. The temperature data was fuzzified through the use of membership functions and an output was determined with the help of the rule base and defuzzification of the results of the rules. The next step in the design procedure was to determine the hardware in which to implement the sensors and the controller.

5.9 Hardware

An Altera Cyclone II field programmable gate array (FPGA), model EP2C35F672C6, was used to perform the controller function of the system. The Cyclone II is compatible with the Nios II soft processor core, chosen to provide the processing core necessary to implement this system in hardware. The Cyclone II FPGA was available in the form of an Altera DE2 Development and Education board, shown in Figure 5.11, and is used in this form. The board provided several advantages over using the FPGA in a standalone fashion. The DE2 board had an additional 8MB of dynamic random access memory (DRAM), some of which was used for this design. The board also provided the programming interface, used by the computer to program the FPGA. The liquid crystal display (LCD) on the board was utilized for user feedback. The system was designed to operate independently of the computer used to program the FPGA

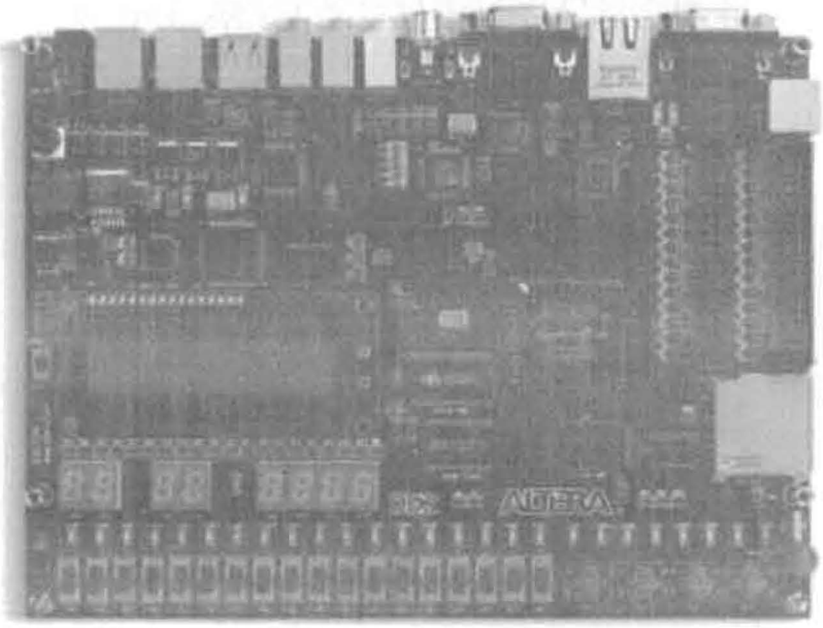

Figure 5.11 Altera DE2.

The temperature was measured with an Analog Devices TMP37 analog temperature sensor. An Analog Devices AD7821 analog to digital converter was used to convert the analog temperature reading into a parallel digital signal.

The heating element was a modified hair dryer. The hair drier was modified so that the high and low settings can be toggled on and off through the use of relays. The cooling elements were simple 80 mm 12 VDC fans. Two fans were used, one blowing air into the enclosure and one acting as an exhaust.

The DE2 board was interfaced to the hair dryer and fans through the use of a switched relay circuit designed and sold by Carl's Electronics. The relay interface provided a buffer between the I/O pins on the FPGA and the relay to protect the FPGA from the relays while they are being switched on and off. This board is shown in Figure 5.12.

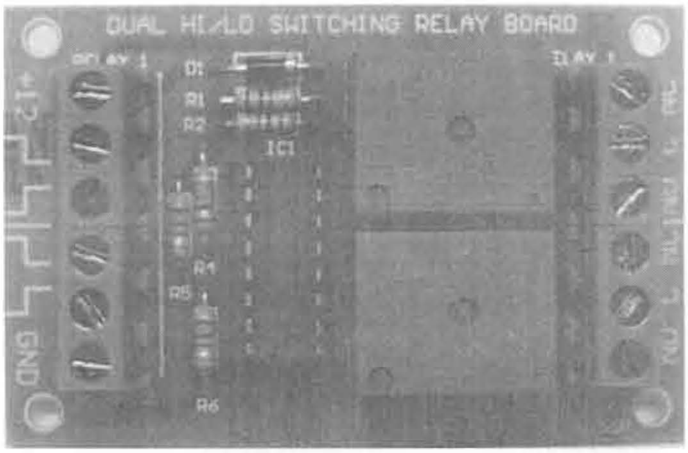

Figure 5.12 Carl's Electronics CK1619 Relay Board.

The power supply was a modified computer ATX power supply, which supplied the 3.3VDC, 5VDC, and 12VDC necessary for the relays, sensors, and other components.

A plastic enclosure containing approximately one cubic foot of air provided the enclosed space. The enclosure was sealed so that air could only enter or exit through the heating and cooling elements. The enclosure was not intentionally insulated to provide a space in which the temperature changes rapidly. The enclosure and all hardware is shown in Figure 5.13.

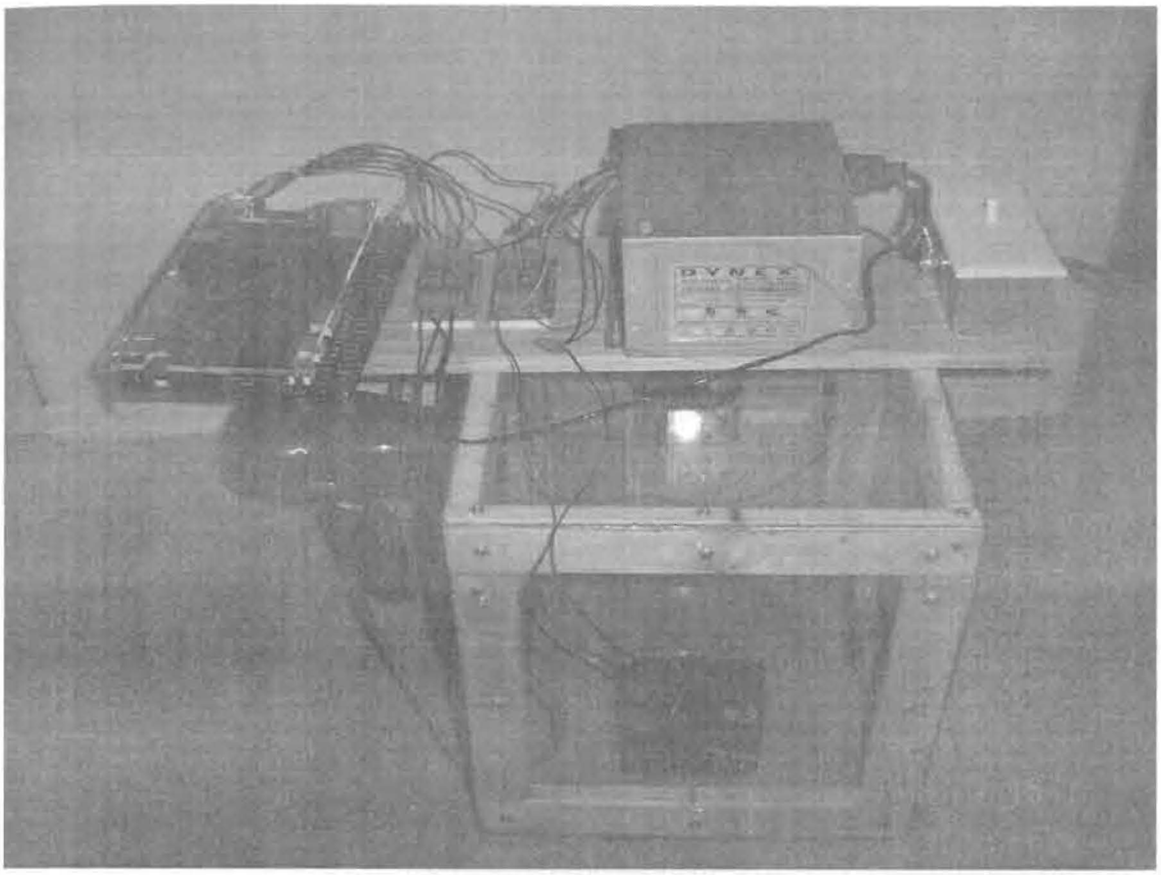

Figure 5.13 Complete System.

5.10 Nios II Processor

The Nios II is a customizable soft processing core created and supported by Altera. The processor can be customized to allow for all components necessary for the design. The nature of this processor makes it ideal for use in fuzzy controllers because it can be easily scaled to work for larger designs. The processing core is created with the system on a programmable chip (SOPC) Builder module of the Quartus II software.

The processing core used for this design was the Nios II/s core. The core features a 32-bit RISC instruction set as well as branch prediction, hardware multiplication, hardware division, and an instruction cache.

The synchronous dynamic random access memory (SDRAM) was accessed by an SDRAM controller integrated into the processor. The controller was designed to access

8MB of SDRAM with 16-bit data width and 12-bit address width. The clock signal for the memory was generated with a phase locked loop (PLL) in the processor core.

A Joint Test Action Group universal asynchronous receiver/transmitter (JTAG UART) was included to interface to the software on the PC and for debugging purposes. No control calculations were performed by the PC.

An LCD controller was used to drive the LCD on the DE2 board. The LCD was used to provide feedback to the user when the controller was run independently of the PC by displaying the current and desired temperature at all times.

A diagram of the micro-controller and outside interfaces is shown in Figure 5.14. The area enclosed in the rectangle represents the components within the FPGA. The components outside of the rectangle are hardware devices interfaced to the controller.

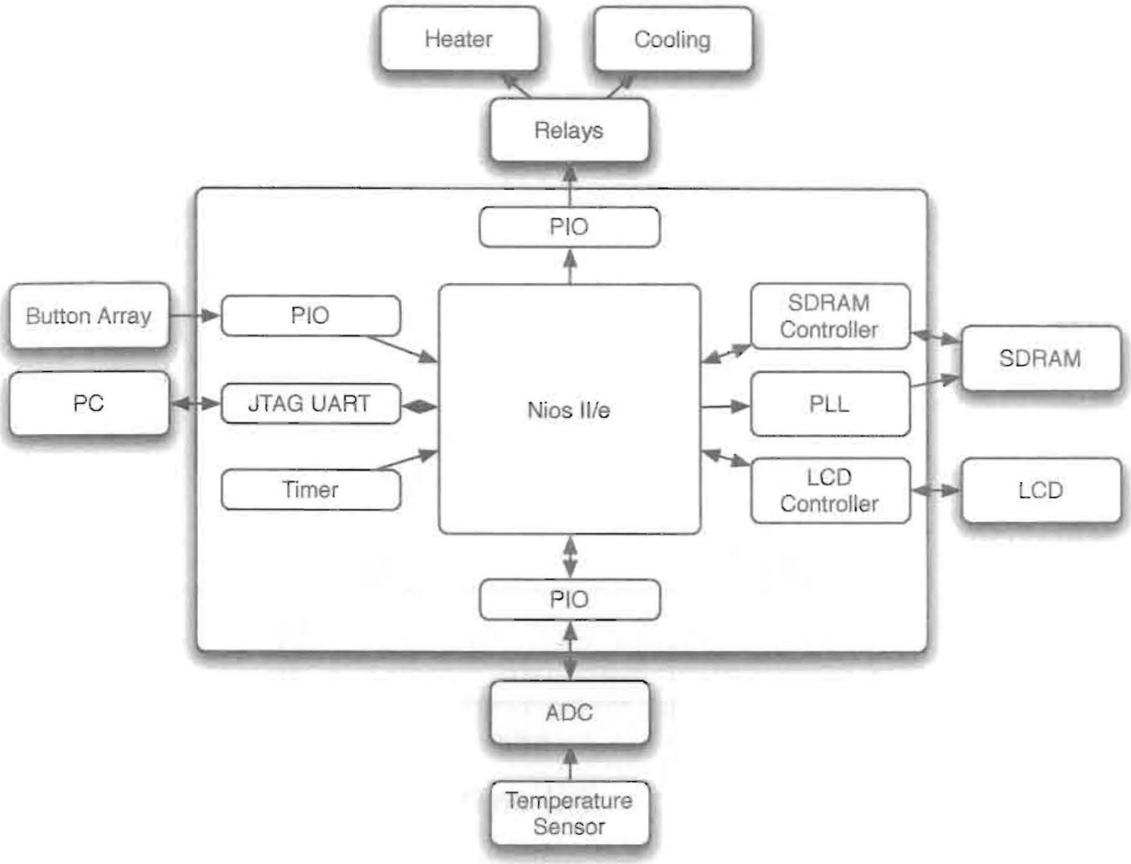

Figure 5.14 Micro-Controller Diagram.

5.11 Experiment Results

In order to test the viability of this control method, an experiment was created to test the controller. The controller was programmed to vary the input signal, *T*, to various temperatures. The output of the system was the current temperature, *t*, and the change in temperature over one sample period, *dt*. The output was transferred to the PC console through the JTAG UART and recorded in order to be plotted. The plot of *T* and *t* versus time can be seen in Figure 5.15

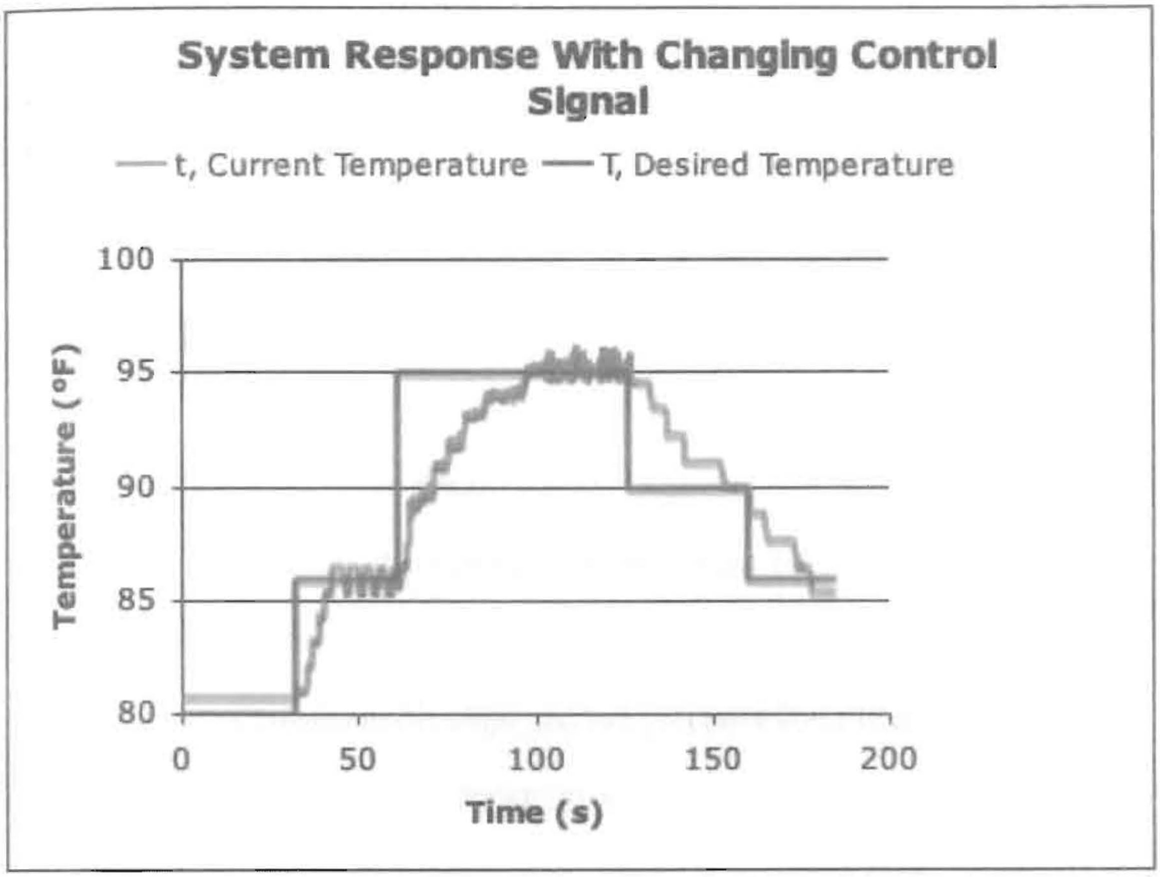

Figure 5.15 System Response at Various Desired Temperatures.

5.12 Conclusions and Future Research

The experiment performed in this work has proven that a controller based on the principals of fuzzy sets and fuzzy logic is a viable alternative to traditional control schemes. Engineers designing simple systems or systems in which the plant is not easily modeled can and should consider implementing a a fuzzy-based controller to simplify the design procedure. The experiment has also shown that the use of FPGA and soft processing technologies provide a good platform on which to design the control system. The use of reprogrammable FPGAs and software based processing cores allows for maximum modularity of fuzzy controllers and is an inexpensive alternative to traditional control system designs. The benefits of using such a system are decreased design lead time, modularity of controllers, basic knowledge of the plant, and more intuitive programming.

The topics of stability and optimality have not been discussed in this work due to the fact that these topics do not have standard principals agreed upon by the scientific community. The areas of stability and optimality of fuzzy based control systems is an area which could benefit from future research. Analytical and experimental research in these areas will strengthen the viability of fuzzy control being used on a much larger scale and in more critical systems.

The use of programmable chips and soft processor cores in control systems also warrants future research. The design of generic controllers which can be easily reprogrammed to suit different plants is a research topic which could be beneficial to industries using many different controllers for similar processes. The nature of the FPGA and soft processing cores make them an ideal foundation for easily reprogrammable and modular controllers. The benefits to industry would consist of decreased cost of controllers, modular replacements, quicker design, and easy reprogramming.

The research conducted in this work also has an application in the classroom. FPGA programming is typically taught as part of courses on digital design. The ability to make a control system from an FPGA may bridge the gap between digital design courses and control system courses. Experiments, such as the one performed in this work, can be constructed to provide students with the ability to take what they have learned in a digital design course and apply it towards what has been learned in a control system course.

Overall, the benefits and increased use of fuzzy controllers ensure that they will continue to be an alternative to traditional control systems for a long time.

Appendices

Appendix A

Classical Set Theory

A.1 Introduction

The material in this section borrows heavily from Chen and Klir [6,7,8].

Set theory is a branch of mathematics dealing with collections of objects. The term *set* refers to any collection of distinct objects that can be grouped into a whole. The collection of all potential values is referred to as the *universe of discourse*. Classical set theory defines a crisp and unambiguous boundary between elements that are members of a set and elements that are not members of a set. There is no method of classifying elements that are only partially contained in a set.

A.2 Notation and Operations

Consider a universe of discourse, $\mathbf{X}$, containing all values that have a particular characteristic. Individual elements in $\mathbf{X}$ are referred to as x. The elements, x, of a universe of discourse are grouped into subsets $\{X, Y, \ldots\}$ that contain elements, x. To indicate that X is a subset of $\mathbf{X}$, write

$$X \subset \mathbf{X}. \tag{A.1}$$

If x is a member of X, write

$$x \in X. \tag{A.2}$$

If x is not a member of X, write

$$x \notin X. \tag{A.3}$$

Sets with no members are referred to as null sets and denoted by $\varnothing$.

Let A and B be two subsets of the universe of discourse. If all members of A are also members of B then A is a subset of B then write

$$A \subset B. \tag{A.4}$$

If all members of B are also members of A, write

$$A=B. \tag{A.5}$$

A subset, A, with specified members, a, and properties, $P_1, P_2, \dots, P_n$, is denoted

$$A = \{a \mid P_1, P_2, \dots, P_n\}. \tag{A.6}$$

The difference of two subsets is defined by

$$A - B = \{a \mid a \in A \text{ and } a \notin B\}. \tag{A.7}$$

Subsets of the universe of discourse are generally referred to as sets if the universe of discourse is implied or inconsequential.

If $\mathbf{X}$ is the universe of discourse then $\mathbf{X}$-A is referred to as the complement of A and is denoted $\overline{A}$.

Some notable properties of complements are

$$\overline{\mathbf{X}} = \varnothing \text{ , } \overline{\varnothing} = \mathbf{X}, \text{ and } \overline{\overline{A}} = A. \tag{A.8}$$

Let $a \in A$ and be real. Then the multiplication of a real scalar value, r, and A be defined as

$$rA = \{ra \mid a \in A\}. \tag{A.9}$$

The union of two subsets (A, B) is defined to be the the set of all elements that are elements of either subset or

$$A \cup B = B \cup A = \{x \mid x \in A \text{ or } x \in B\}. \tag{A.10}$$

The intersection of two subsets (*A*,*B*) is defined to be the set of all elements that are elements of both *A* and *B* or

$$A \cap B = B \cap A = \{x \mid x \in A \text{ and } x \in B\}. \qquad \text{(A.11)}$$

Let *A*, *B* and *C* be subsets of the universe of discourse, **X**. The following properties are valid for all sets:

$$\overline{\overline{A}} = A \qquad \text{(Involutive Law)} \qquad \text{(A.12)}$$

$$A \cup B = B \cup A \qquad \text{(Commutative Law)} \qquad \text{(A.13)}$$

$$A \cap B = B \cap A \qquad \text{(A.14)}$$

$$(A \cup B) \cup C = A \cup (B \cup C) \qquad \text{(Associative Law)} \qquad \text{(A.15)}$$

$$(A \cap B) \cap C = A \cap (B \cap C) \qquad \text{(A.16)}$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \qquad \text{(Distributive Law)} \qquad \text{(A.17)}$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \qquad \text{(A.18)}$$

$$A \cup A = A \qquad \text{(A.19)}$$

$$A \cap A = A \qquad \text{(A.20)}$$

$$A \cup (A \cap B) = A \qquad \text{(A.21)}$$

$$A \cap (A \cup B) = A \qquad \text{(A.22)}$$

$$A \cup (\overline{A} \cap B) = A \cup B \qquad \text{(A.23)}$$

$$A \cap (\overline{A} \cup B) = A \cap B \qquad \text{(A.24)}$$

$$A \cup \mathbf{X} = \mathbf{X} \qquad \text{(A.25)}$$

$$A \cap \mathbf{X} = A \qquad \text{(A.26)}$$

$$A \cap \varnothing = \varnothing \qquad \text{(A.27)}$$

$$A \cup \varnothing = A \tag{A.28}$$

$$A \cap \overline{A} = \varnothing \tag{A.29}$$

$$A \cup \overline{A} = \mathbf{X} \tag{A.30}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B} \qquad \text{(DeMorgan's Law)} \tag{A.31}$$

$$\overline{A \cup B} = \overline{A} \cap \overline{B} \tag{A.32}$$

Appendix B

Classical Logic

B.1 Introduction

The material in this section borrows heavily from Chen and Klir [6,7].

Classical logic deals with propositions or, more specifically, the truth or falsity of those propositions. A proposition, P, is a set of elements of which each member of the set is either denoted as true or false. The set of true values is referred to as the truth set and is denoted T(P). The false values are referred to as the falsity set. As with classical sets, the boundary between the truth set and the falsity set is crisp and unambiguous. There can be no elements that are partly true or partly false.

B.2 Notation and Operations

Logical expressions are formed with five basic connectives. Propositions are combined using the following connectives.

$(\vee)$	Disjunction		(B.1)
$(\wedge)$	Conjunction		(B.2)
$(-)$	Negation		(B.3)
$(\rightarrow)$	Implication	(antecedent $\rightarrow$ consequent)	(B.4)
$(\leftrightarrow)$	Equivalence		(B.5)

Let P and Q be two propositions on the same universe of discourse.

The disjunctive connective is similar to the logic *or* and is more commonly referred to as the *inclusive or.* A compound proposition is true if either or both of the simple propositions are true.

The conjunctive connective is similar to the logic *and.* A compound proposition is true if and only if both simple propositions are true. Table B.1 shows the truth table for the conjunctive connective.

Negation makes a member of the truth subset a member of the falsity set or a member of the falsity subset a member of the truth subset. Negation is similar to logic *not.* The truth table representing negation is shown in Table B.2.

The implicative connective means that the antecedent implies the consequent. If the first proposition is true then the second proposition is also true. If the antecedent is false, no information can be determined from it and it can imply truth or falsity of the consequent as shown in the truth table in Table B.3.

The equivalent connective means that both propositions are members of the same truth or falsity set. The truth table for the equivalent connective is shown in Table B.4

Table B.1 Conjunctive Truth Table

P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F

Table B.2 Negation Truth Table

P	¬P
T	F
F	T

Table B.3 Implicative Truth Table

P	Q	P⇒Q
T	T	T
F	T	T
T	F	F
F	F	T

Table B.4 Equivalent Truth Table

P	Q	P⇔Q
T	T	T
F	T	F
T	F	F
F	F	T

Appendix C

Source Code

C.1 Main Function

The code in this section initializes the variables and devices used to implement the fuzzy controller. The main program loop is also contained in this section of code. The main program loop reads the temperature from the sensor, fuzzifies the input, applies the rule base, and defuzzifies the output. The loop is infinite.

```
/*************************************************************************
 * File name: fuzzy_control.c
 * Description: Implements fuzzy controller.
 * Author: Eric Stauffer
 * References: Altera provided source code.
 * Notes: None.
*************************************************************************/

#include "fuzz_functions.h"
#include "decs_and_vars.h"
#include "relay_functions.h"
#include "lcd_and_buttons.h"
#include "temp_sensor.h"

/*************************************************************************
 * Function name: main
 * Description: Main function and infinite loop.
 * Input: Void.
 * Output: Void.
 * Notes: See comments.
*************************************************************************/

int main(void)
{
 //Fuzzy membership values.
 float hot, warm, cold, cool, neutral, heating, cooling, constant_t;
```

```
//Firing strength of each rule.
 float rule[18];

 //Maximum rule value.
 float max_value=0;

 //Used to average temperature readings.
 float aggregate=0;

 //Previous temperature reading.
 float prev_temp;

 //Index corresponding to maximum rule firing strength.
 int max_index=0;

 //Indices.
 int i,d;

 //LCD address.
 FILE * lcd;

 //Set initial value of desired temperature.
 des_temp=85;

//Ensure all relays are off.
 all_off();

 //Initialize the ADC.
 IOWR_ALTERA_AVALON_PIO_DATA(TEMP_CTRL_PIO_BASE,0x1);

 //Get address of LCD.
 lcd = fopen("/dev/lcd_display", "w");

 //Initialize the LCD.
 lcd_init( lcd );

 //initialize the button PIO.
 init_button_pio();

 //Read initial temperature.
 cur_temp=read_temp();

 //Enter infinite loop.
```

```
while( 1 )
{

 //Current temperature reading becomes previous temperature.
 prev_temp=cur_temp;

 //Take 25 temperature readings and average them to eliminate bad readings.
 for(d=0;d<25;d++)
  {
  aggregate=read_temp()+aggregate;
  }

 //Current temperature is average of 25 temperature readings.
 cur_temp=aggregate/25;

 //Reset aggregate variable.
 aggregate=0;

 //Print current temperature to PC for analysis.
 printf("%f ",cur_temp);

 //Check for button presses and react. Update LCD.
 handle_button_press(lcd);

 /*The following functions determine the membership values in each of the
  * fuzzy subsets.
  */
 heating=fuzz_heating(cur_temp,prev_temp);
 constant_t=fuzz_constant_t(cur_temp,prev_temp);
 cooling=fuzz_cooling(cur_temp,prev_temp);
 hot = fuzz_hot(cur_temp);
 warm = fuzz_warm(cur_temp);
 neutral = fuzz_neut(cur_temp);
 cool = fuzz_cool(cur_temp);
 cold = fuzz_cold(cur_temp);

 /*The following functions calculate the firing strength of every rule
  * in the fuzzy control system.
  */
 rule[0]=min(heating,hot);
 rule[1]=min(heating,warm);
 rule[2]=min(heating,neutral);
 rule[3]=min(heating,cool);
```

```
rule[4]=min(heating,cold);
rule[5]=min(constant_t,hot);
rule[6]=min(constant_t,warm);
rule[7]=min(constant_t,neutral);
rule[8]=min(constant_t,cool);
rule[9]=min(constant_t,cold);
rule[10]=min(cooling,hot);
rule[11]=min(cooling,warm);
rule[12]=min(cooling,neutral);
rule[13]=min(cooling,cool);
rule[14]=min(cooling,cold);

/*The following functions perform the defuzzification of the output by
 * determining the maximum firing strength and corresponding rule. The
 * switch determines the correct output state based upon the rule with the
 * maximum firing strength.
 */
for (i=0;i<15;i++){
 if (max_value<=rule[i]){
  max_value=rule[i];
  max_index=i;
 }
}

switch (max_index)
{
 case 0:
  high_cool();
  break;

 case 1:
  high_cool();
  break;

 case 2:
  all_off();
  break;

 case 3:
  low_heat();
  break;

 case 4:
```

```
 high_heat();
 break;

case 5:
 high_cool();
 break;

case 6:
 low_cool();
 break;

case 7:
 all_off();
 break;

case 8:
 low_heat();
 break;

case 9:
 high_heat();
 break;

case 10:
 high_cool();
 break;

case 11:
 low_cool();
 break;

case 12:
 all_off();
 break;

case 13:
 high_heat();
 break;

case 14:
 high_heat();
 break;
```

```
      default:
        break;

    }

    //Reset variables.
    max_index=0;
    max_value=0;
    };

  fclose(lcd);
  return 0;
}
```

C.2 Declarations and Variables

The code in this section defines all of the libraries needed for the compiler as well as the global variables used to store the terminal commands for the LCD. There is no code executed in this section. Some of the code was borrowed from Altera-supplied source code and the Altera license agreement is included at their request.

```
/**********************************************************************
 * File name: decs_and_vars.h
 * Description: Include statements, function declarations, and global variables.
 * Author: Eric Stauffer and Altera
 * References: Altera source code.
 * Notes: See license agreement.
**********************************************************************/

#include "alt_types.h"
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"

//Edge capture pointer variable.
volatile int edge_capture;

//Desired temperature and current temperature are global variables.
float des_temp,cur_temp;

//Function declarations
void high_heat (void);
void low_heat (void);
void high_cool (void);
void low_cool (void);
void all_off (void);
float fuzz_warm(float);
float fuzz_hot(float);
float fuzz_neut(float);
float fuzz_cool(float);
float fuzz_cold (float);
float fuzz_heating(float, float);
float fuzz_cooling(float, float);
float fuzz_constant_t(float, float);
float min(float, float);
float read_temp(void);

/*The following code was provided by Altera and is used to set the cursor
 * position on the LCD display. All codes are VT100 terminal compliant.
 */

/* Integer ASCII value of the ESC character. */
#define ESC 27

/* Position cursor at row 1, column 1 of LCD. */
#define ESC_TOP_LEFT "[1;1H"

/* Position cursor at row2, column 5 of LCD. */
#define ESC_COL2_LEFT "[2;1H"

/* Clear */
#define ESC_CLEAR "K"

/* Clear all */
#define ESC_CLEAR_ALL "[ 2 J"

/* Position cursor at row1, column 5 of LCD. */
#define ESC_COL1_INDENT5 "[1;5H"
```

C.3 Membership Functions

The code in this section implements the membership functions as described in Chapter 5. There is one function for each membership function, depicted in Figures 5.6 and 5.10. Each section of code implements a piecewise linear frunction as shown in Equations 5.1-5.8.

```
/*********************************************************************
 * File name: fuzz_functions.h
 * Description: Implements membership functions.
 * Author: Eric Stauffer
 * References: None.
 * Notes: None..
 *********************************************************************/

#include "decs_and_vars.h"


/**********************************************************************
 * Function name: fuzz_warm
 * Description: Implements "warm" membership function.
 * Input: Current temperature.
 * Output: Membership value in "warm" fuzzy subset.
 * Notes: N/A.
 **********************************************************************/
float fuzz_warm(float temp)
{
 float i;
 if (temp==(des_temp+3)) {
  return 1;
 } else if(temp>=(des_temp+6)||(temp<=(des_temp))) {
  return 0;
 } else if((temp>des_temp)&&(temp<(des_temp+3))){
  i = (temp-des_temp)/3;
  return i;
 } else if((temp>(des_temp+3))&&(temp<(des_temp+6))) {
  i=(-temp+des_temp+6)/3;
  return i;
 };
 return -1;
}


/*******************************************************************
 * Function name: fuzz_hot
 * Description: Implements "hot" membership function.
 * Input: Current temperature.
 * Output: Membership value in "hot" fuzzy subset.
 * Notes: N/A.
*********************************************************************/
float fuzz_hot(float temp)
{
 float i;
 if (temp>=(des_temp+6)) {
  return 1;
 } else if(temp<=(des_temp+3)) {
  return 0;
 } else if(temp>(des_temp+3)&&temp<(des_temp+6)){
  i = (-temp+des_temp+6)/3;
  return i;
 };
 return -1;
}


/*******************************************************************
 * Function name: fuzz_neut
 * Description: Implements "neutral" membership function.
 * Input: Current temperature.
 * Output: Membership value in "neutral" fuzzy subset.
 * Notes: N/A.
 * *****************************************************************/
float fuzz_neut(float temp)
{
 float i;
 if (temp==des_temp) {
  return 1;
 } else if(temp<=(des_temp-3)||(temp>=(des_temp+3))) {
  return 0;
 } else if((temp<des_temp)&&(temp>(des_temp-3))){
  i = (temp-des_temp+3)/3;
```

```
   return i;
  } else if((temp>des_temp)&&(temp<(des_temp+3))) {
   i=(-temp+des_temp+3)/3;
   return i;
  };
 return -1;
}


/************************************************************************
 * Function name: fuzz_cool
 * Description: Implements "cool" membership function.
 * Input: Current temperature.
 * Output: Membership value in "cool" fuzzy subset.
 * Notes: N/A.
 ************************************************************************/
float fuzz_cool(float temp)
{
 float i;
 if (temp==(des_temp-3)) {
 return 1;
 } else if (temp<=(des_temp-6)||(temp>=(des_temp))) {
   return 0;
 } else if ((temp>(des_temp-6))&&(temp<(des_temp-3))){
   i = (temp-des_temp+6)/3;
   return i;
 } else if((temp>(des_temp-3))&&(temp<(des_temp))) {
   i=(-temp+des_temp)/3;
   return i;
 };
 return -1;
}


/************************************************************************
 * Function name: fuzz_cold
 * Description: Implements "cold" membership function.
 * Input: Current temperature.
 * Output: Membership value in "cold" fuzzy subset.
 * Notes: N/A.
\***********************************************************************/
float fuzz_cold (float temp)
{
 float i;
 if (temp<=(des_temp-6)) {
   return 1;
 }else if(temp>=(des_temp-3)) {
   return 0;
 }else if(temp<(des_temp-3)&&temp>(des_temp-6)){
   i = (-temp+des_temp-3)/3;
   return i;
 };
 return -1;
}


/***********************************************************************
 * Function name: fuzz_heating
 * Description: Implements "heating" membership function.
 * Input: Current temperature and previous temperature.
 * Output: Membership value in "heating" fuzzy subset.
 * Notes: N/A.
***********************************************************************/
float fuzz_heating(float temp, float prev_temp)
{
 float tc;
 tc=temp-prev_temp;
 if (tc<=0){
   return 0;
 }else if (tc>=1){
   return 1;
 }else if(tc<1&&tc>0){
   return tc;
 }
 return -1;
}


/***********************************************************************
 * Function name: fuzz_cooling
 * Description: Implements "cooling" membership function.
 * Input: Current temperature and previous temperature.
 * Output: Membership value in "cooling" fuzzy subset.
 * Notes: N/A.
***********************************************************************/
float fuzz_cooling(float temp, float prev_temp)
```

```
{
 float tc;
 tc=temp-prev_temp;
 if (tc<=-1){
  return 1;
 }else if (tc>=0){
  return 0;
 }else if(tc>-1&&tc<0){
  return -tc;
 }
 return -1;
}


/**********************************************************************
 * Function name: fuzz_constant_t
 * Description: Implements "constant" membership function.
 * Input: Current temperature and previous temperature.
 * Output: Membership value in "constant" fuzzy subset.
 * Notes: N/A.
**********************************************************************/
float fuzz_constant_t(float temp, float prev_temp)
{
 float tc;
 tc=temp-prev_temp;
 if (tc==0){
  return 1;
 }else if (tc<=-1||tc>=1){
  return 0;
 }else if(tc>-1&&tc<0){
  return tc+1;
 }else if(tc>0&&tc<1){
  return 1-tc;
 }
 return -1;
}


/**********************************************************************
 * Function name: min
 * Description: Returns the minimum of the two values passed to the function.
 * Input: Two arguments.
 * Output: The lower-valued argument.
 * Notes: N/A.
***********************************************************************/
float min(float arg1, float arg2)
{
 if (arg1>arg2){
  return arg2;
 }else if(arg2>arg1){
  return arg1;
 }else if (arg2==arg1){
  return arg2;
 };
 return -1;
}
```

C.4 LCD and Buttons

The code in this section contains the functions necessary to initialize the LCD and the button array. The button interrupt handling function is also included in this section.

```
/************************************************************************
 * File name: lcd_and_buttons.h
 * Description: Functions for controlling the LCD and reading data from the
 *              button array.
 * Author: Altera, modified by Eric Stauffer
 * References: Altera code provided with count_binary application.
 * Notes: See licence agreement.
 ************************************************************************/

#include "decs_and_vars.h"

/************************************************************************
 * Function name: handle_button_interrupts
 * Description: Processes button presses and associated interrupts
 * Input: Data from interrupt register.
 * Output: None.
 * Notes: N/A.
 ************************************************************************/
static void handle_button_interrupts(void* context, alt_u32 id)
{
  /* Cast context to edge_capture's type. It is important that this be
   * declared volatile to avoid unwanted compiler optimization.
   */
  volatile int* edge_capture_ptr = (volatile int*) context;
  /* Store the value in the Button's edge capture register in *context. */
  *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP
(BUTTON_PIO_BASE);
  /* Reset the Button's edge capture register. */
  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0);
}

/************************************************************************
 * Function name: init_button_pio
 * Description: Initialization of the button PIO and edge capture register.
 * Input: None.
 * Output: None.
 * Notes: None.
***************************************************************************/
static void init_button_pio(void)
{
 /* Recast the edge_capture pointer to match the alt_irq_register() function
  * prototype. */
 void* edge_capture_ptr = (void*) &edge_capture;
 /* Enable all 4 button interrupts. */
 IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE, 0xf);
 /* Reset the edge capture register. */
 IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0x0);
 /* Register the interrupt handler. */
 alt_irq_register( BUTTON_PIO_IRQ, edge_capture_ptr, handle_button_interrupts );
}

/***************************************************************************
 * Function name: lcd_init
 * Description: Initializes the LCD.
 * Input: LCD address.
 * Output: None.
 * Notes: None.
***************************************************************************/
static void lcd_init( FILE *lcd )
{
 fprintf(lcd, "%c%s DESIRED CURRENT", ESC, ESC_TOP_LEFT);
}

/***************************************************************************
*  Function name: update_lcd
 * Description: Displays updated temperature date on LCD.
 * Input: LCD address.
 * Output: None.
 * Notes: None.
***************************************************************************/
static void update_lcd( void* arg )
{
 FILE *lcd = (FILE*) arg;
 fprintf(lcd, "%c%s  %.1f   %.1f", ESC, ESC_COL2_LEFT, des_temp, cur_temp);
}

/***************************************************************************
 * Function name: handle_button_press
 * Description: Defines action to take upon button press.
 * Input: LCD address.
 * Output: None.
 * Notes: Some debugging framework left, code removed.
***********************************************************************/
static void handle_button_press(FILE *lcd)
{
 switch (edge_capture)
 {
  case 0x1:
   des_temp=des_temp+1;
   break;
  case 0x2:
   des_temp=des_temp-1;
   break;
  case 0x4:
   //Used for debugging.
   break;
  case 0x8:
   //Used for debugging.
   break;
  default:
   break;
  }
 //Reset the edge capturer register.
 edge_capture = 0;
 //Update the temperature data on the LCD.
 update_lcd(lcd);
}
```

C.5 Relay Functions

The code in this section toggles the relays, depicted in Figure 5.12.

```
/*************************************************************************
 * File name: relay_functions.h
 * Description: Functions for toggling relays.
 * Author: Eric Stauffer
 * References: None.
 * Notes: None..
*************************************************************************/

#include "decs_and_vars.h"

/*************************************************************************
 * Function name: high_heat
 * Description: Turns on the relay which controls the high heat setting.
 * Input: Void.
 * Output: Void.
 * Notes: N/A.
*************************************************************************/
void high_heat (void)
{
 IOWR_ALTERA_AVALON_PIO_DATA(RELAY_PIO_BASE, 0x8);
}


/*************************************************************************
 * Function name: low_heat
 * Description: Turns on the relay which controls the low heat setting.
 * Input: Void.
 * Output: Void.
 * Notes: N/A.
 *
*************************************************************************/
void low_heat (void)
{
 IOWR_ALTERA_AVALON_PIO_DATA(RELAY_PIO_BASE, 0x4);
}
```

```
/**************************************************************************
 * Function name: high_cool
 * Description: Turns on the relays which control the high cool setting.
 * Input: Void.
 * Output: Void.
 * Notes: N/A.
**************************************************************************/
void high_cool (void)
{
 IOWR_ALTERA_AVALON_PIO_DATA(RELAY_PIO_BASE, 0x3);
}

/**************************************************************************
 * Function name: low_cool
 * Description: Turns on the relay which controls the low cool setting.
 * Input: Void.
 * Output: Void.
 * Notes: N/A.
**************************************************************************/
void low_cool (void)
{
 IOWR_ALTERA_AVALON_PIO_DATA(RELAY_PIO_BASE, 0x1);
}

/**************************************************************************
 * Function name: all_off
 * Description: Turns off all relays.
 * Input: Void.
 * Output: Void.
 * Notes: N/A.
**************************************************************************/
void all_off (void)
{
 IOWR_ALTERA_AVALON_PIO_DATA(RELAY_PIO_BASE, 0x0);
}
```

C.6 Temperature Readings

The code in this section contains functions for reading temperature from the analog to digital converter. Code is also included to convert the temperature read in binary format from the analog to digital converter into decimal numbers. The decimal numbers are converted to °F.

```
/************************************************************************
 * File name: temp_sensor.h
 * Description: Implements code necessary to communicate with temperature sensor.
 * Author: Eric Stauffer
 * References: None.
 * Notes: None..
 ************************************************************************/

#include "decs_and_vars.h"

/************************************************************************
 * Function name: read_temp
 * Description: Reads the current temperature from the a-to-d converter.
 * Input: Void.
 * Output: Current temperature.
 * Notes: See comments.
 ************************************************************************/
float read_temp(void)
{
 /*"meas_val" is an array which stores the individual bits of the
  * temperature reading. "steps" is an aggregate used to determine the decimal
  * value of the temperature reading. "tempbin" is the binary representation
  * of the raw data read form the a-to-d converter. "temp" is the temperature
  * in degrees fahrenheit.  */
 alt_u8 tempbin;
 int meas_val[8];
 int steps=0;
 float temp;
 int i;

 //Send a read request to the a-to-d converter.
 IOWR_ALTERA_AVALON_PIO_DATA(TEMP_CTRL_PIO_BASE,0x1);
IOWR_ALTERA_AVALON_PIO_DATA(TEMP_CTRL_PIO_BASE,0x0);
IOWR_ALTERA_AVALON_PIO_DATA(TEMP_CTRL_PIO_BASE,0x1);
//Wait at least 530ns for the data to be ready to read.
usleep(15);
//Read the data from the a-to-d converter.
tempbin=IORD_ALTERA_AVALON_PIO_DATA(TEMP_DATA_PIO_BASE);

//Decompose the raw data to individual bits to convert to decimal.
meas_val[0]=(tempbin&0x01);
meas_val[1]=(tempbin&0x02);
meas_val[2]=(tempbin&0x04);
meas_val[3]=(tempbin&0x08);
meas_val[4]=(tempbin&0x10);
meas_val[5]=(tempbin&0x20);
meas_val[6]=(tempbin&0x40);
meas_val[7]=(tempbin&0x80);

//Convert raw binary data to decimal number.
for (i=7;i>=0;i--)
  {
   steps=steps+meas_val[i];
  }

//Convert decimal value to degrees fahrenheit.
temp=32+(1.8*steps*.01289)/(.02);

//Return the decimal value.
return temp;
}
```

Appendix D

Altera Programming Tools

D.1 Introduction

The Altera DE2 comes packaged with all of the programming tools necessary to create the controller system used in this work. Altera provides the Quartus II Web Version and the Nios II IDE software with the purchase of the DE2 board. Quartus II is used to design the logic circuits being programmed on the FPGA. The Nios II IDE software is used to program the microprocessor created with Quartus II and SOPC Builder. This appendix provides an introduction to both applications.

D.2 Quartus II

All designs typically begin in the Quartus II application. This application allows the user to create the logic circuits to be programmed on the FPGA. Since the FPGA begins as a virtual blank slate, the designing engineer must specify connections to each and every pin being utilized in the design. The only exception is the power supply pins, because the board has the power pins hardwired to the power supply. Quartus II allows schematic entry in several forms such as VHDL, Verilog, and direct schematic entry with the block diagram editor. Quartus II is also the home of the SOPC Builder module used to design the Nios II processor. The rest of this section will provide an overview and brief tutorial of the Quartus II application. The next section will provide an overview of the SOPC Builder module.

To begin a new project in Quartus II, open the application and click “File” and then “New Project Wizard”. The wizard will open and prompt the user for the name of the project, the directory in which to store the project, and the name of the top level design. The top level design file name must be assigned properly or segments of code may not be compiled. The wizard will then proceed to prompt for any additional files and also for the hardware that the design will be implemented on. Clicking on "Finish" will open the new project in the main Quartus II window.

At this point, the user may begin to create their design. For this example, a small VHDL file will be used. To create the VHDL file, click on "File" and then click on "New". This will open the dialog box prompting the user for the type of file to be created. Since this design will be a simple VHDL file, select VHDL and click "OK". A blank VHDL file is now available. Since this is a simple design, only one file will be necessary so this VHDL file will become the top level design file by saving as "test.vhd". At this point, standard VHDL code may be entered and the following code will be used for this tutorial:

```
library ieee;
use ieee.std_logic_1164.all;
entity and_gate is
port ( x,y        :in std_logic;
          z        :out std_logic);
end and_gate;
architecture a_gate of and_gate is
begin
z <= x and y;
end a_gate;
```

The code describes a very simple AND gate with two inputs and one output. The next step in the design process is to interface this code to the outside world by assigning pins to each of the two inputs and the output. Assign pins by clicking on "Assignments" and then “Assignment Editor”. The VHDL variables are placed in the “From” column and the pin name is placed in the “To” column.

The input pins can be tied to any of the other devices on the board or assigned to the I/O headers for connection to outside devices. For this tutorial, assign the two input pins to the logic switches and the output pin to one of the LEDs.

At this point, the design is ready to be compiled and transferred to the DE2 board for testing. To compile the design, click on “Processing” and then “Start Compilation”. After compilation, click on “Tools” and then “Programmer”. The programmer utility will transfer the design to the DE2 board for further testing.

It should be noted that this is only a very brief tutorial and the Quartus II software is a very exhaustive design suite with the ability to verify timing, run simulations, and modify board signal routing, among other things. The tutorial creates a working design but certainly does not showcase most of the features that the software is capable of. The next section provides an introduction to one of these tools, the SOPC Builder.

D.3 SOPC Builder

The SOPC Builder is used to create a System on a Programmable Chip, or SOPC design. This module is used to implement the Nios II processor as well as the interfaces and controllers for some other devices that the FPGA may interface with. In the example system created in Chapter 5, the SOPC Builder was used to create the processor, memory controller, phase locked loop, parallel input/output, and PC interface. This section provides a brief introduction.

The SOPC Builder module is part of the Quartus II software package. To access SOPC Builder, open Quartus II, create a new project, and click on "File", "New", and then "SOPC Builder System". Name the system, if the entire design will be done in SOPC Builder the name of the system should be the same as the top level design entry. Also, choose an HDL; either VHDL or Verilog. At this point, elements of the design can be double-clicked to be added to the system. When added, a dialog box will typically appear with options that are available for that component.

Once all desired components have been added, click on "Generate" to generate the HDL code that describes the system. After generation, the system is available in Quartus II to complete the design. Supplemental code may be added in Quartus II to interface to the system or the system may be interfaced to the outside world with the use of pin assignments and the I/O headers.

It must be noted that before the pins can be assigned, the "Analysis and Synthesis" steps of the compilation must be completed by clicking on "Processing", "Start", and then "Start Analysis & Synthesis". After this step, the pins are available to be assigned in the pin assignment editor. If a Nios II processor is implemented in the SOPC Builder, the code for that processor may be created and tested in the Nios II IDE program, also bundled with the DE2 board, and the topic of the next section.

D.4 Nios II IDE

The Nios II IDE application is designed to cross-compile from C/C++ to Nios II assembly language. The application requires information about the processor compiled in the SOPC Builder in order to configure the I/O addresses and memory ranges. Load the program and click on "File", "New", and "C/C++ Application" and then choose the .ptf file generated by Quartus II, this will provide all of the information used to generate the system. Once all of the C/C++ code has been entered, right click on the project folder in the navigator window. Select "Run" and then "As Nios II Hardware". The code will be compiled, transferred to the board, and executed.

Most C/C++ commands are supported natively in the Nios II IDE application and header files and libraries may be added for maximum design flexibility. The design environment is very feature rich, including features such as step-by-step execution and debugging as well as a complete instruction set simulator to simulate software without the benefit of a hardware device to program and test. The application provides a very integrated design environment with all features necessary to compile and test code very effectively.

References

[1] Ross, Timothy J. Fuzzy Logic with Engineering Applications, Second Edition. England: John Wiley & Sons Ltd, 2004.

[2] Valvano, Jonathan W. Embedded Microcomputer Systems Real Time Interfacing. Pacific Grove: Brooks / Cole, 2000.

[3] Cox, E. The Fuzzy Systems Handbook. Cambridge: Academic Press Professional, 1994.

[4] Geobel, Greg. An Introduction to Fuzzy Control Systems. 2/2/2006. <http://www.faqs.org/docs/fuzzy/>

[5] Kosko, B. Fuzzy Engineering. Upper Saddle River: Prentice-Hall, 1997.

[6] Chen, Guanrong and Pham, Trung Tat. Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems. Boca Raton: CRC Press LLC, 2001.

[7] Klir, G. and Folger, T. Fuzzy Sets, Uncertainty, and Information. Englewood Cliffs: Prentice Hall, 1988.

[8] Klir, G. and yuan, B. Fuzzy Sets and Fuzzy Logic. Upper Saddle River: Prentice Hall, 1995.

[9] Fuzzy Sets::Tutorial (Fuzzy logic Toolbox) Mathworks. 9/29/2005 9:43PM. <http://www.mathworks.com/access/helpdesk/help/toolbox/fuzzy/>

[10] Understanding Neural Networks and Fuzzy logic : Basic Concepts and Applications. New York: IEEE Press, 1996.

[11] Kosko, B. Neural Networks and Fuzzy Systems. Englewood Cliff: Prentice-Hall, 1992.

[12] Kosko, B. Fuzzy Thinking. New York: Hyperion Press, 1993.

[13] Yurkovich, Stephen and Passino, Kevin M. Fuzzy Control. Menlo Park: Addison Wesley, 1998.

[14] Kaehler, Steven. Fuzzy Logic - An Introduction. 9/16/2005. <http://www.seattlerobotics.org/encoder/mar98/fuz/>

[15] Yurkovich, Stephen and Passino, Kevin M. "A Laboratory Course on Fuzzy Control." IEEE Transactions on Education. 42 (1999): 15-21.

[16] Terano, T., Asai, K., and Sugeno, M. Fuzzy System Theory and its Applications. San Diego: Academic Press, 1992.