Fuzzy Logic Based Automotive Airbag Control System

by
Tariq M. Mian

A Thesis

Submitted to the College of Graduate Studies and Research through the

Department of Electrical and Computer Engineering in Partial Fulfillment

of the Requirements for the Degree of

Master of Applied Science

at the

University of Windsor

Windsor, Ontario, Canada

December 1999.

901972

ABSTRACT

Fuzzy Logic implementation is becoming increasingly important, and finding applications in diverse areas of current interest, such as control, pattern recognition, robotics, and other decision making applications. Fuzzy decision process offer a significant advantage over crisp decision process which is the ability to process different levels of truth instead of only 1 or 0 levels. Fuzzy Logic does not require precise inputs, it is inherently robust, and can process any reasonable number of inputs but system complexity increases rapidly with more inputs and outputs. Distributed processors would probably be easier to implement. Simple, plain-language IF X AND Y THEN Z rules are used to describe the desired system response in terms of linguistic variables rather than mathematical formulas. The number of these is dependent on the number of inputs, outputs, and the designer's control response goals.

The new Motorola 68HC12 MCU has an embedded fuzzy logic instruction set. Using this instruction set, we can implement complex fuzzy logic systems using only a few hundred bytes of ROM that cycle compute in less than a millisecond. Considering the fact that the fuzzy logic instruction set of the 68HC12, enables the use of fuzzy logic in mass-market high-speed applications, such as car engine control, anti-skid brakes, traction control, inter-vehicle dynamics control, hard disk drive control, servo motor control, and cellular phones.

This thesis deals with the design of Automotive Airbag Control System a using Fuzzy Logic based decision structure and implementation using the 68HC12 microcontroller.

To: **My Parents**

My Wife Farah

and my Son Moaaz

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisors Dr. J. J. Soltis and for his encouragement, support and guidance during the course of this research. I would also like to thank the my thesis committee members Dr. M. A. Sid Ahmed, Dr. R. Gaspar and Prof. Philip H. Alexander for their comments and advice.

Finally my warmest acknowledgments to my wife, our parents and our families for their constant love and support, who wished and prayed for my success and their love and support is always a milestone in my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Air bags are safety devices of proven value that supplement the protection provided by seat belts. But air bags sometimes cause injuries when they inflate in low-speed collisions and incidents in which air bags do not inflate when it seemed they should have. Virtually all new cars have airbags, and they're saving lives. They are reducing driver deaths by about 14 percent, and passenger bags reduce deaths by about 11 percent.

1.2 Construction and Design

Airbag assemblies consist of the airbag (made of Nylon), inflator modules, sensor

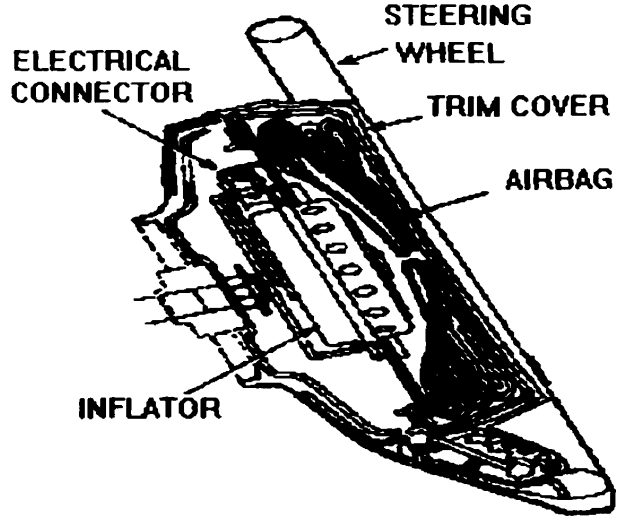

Fig: 1 Airbag Construction

housing, electrical connectors (clock spring), airbag retainer and the cover. The driver's

side bag is mounted in the center of the steering wheel while the passenger bag is mounted in the top of the dash on the passenger side of the vehicle. In addition to the front airbags, the car companies are putting Airbags in the doors for side impacts that are not covered by the primary airbags. They are putting them in the seats for the drivers and rear passengers as well. This increases the cost as well as the complexity of the systems.[1]

1.3 What happens in a Collision

When there is a frontal collision a number of things happen very quickly. The sudden deceleration of the vehicle causes 2 sensors to send an electrical signal to the diagnostic module. The diagnostic module self tests to confirm that a crash event is taking place. then it allows the signal to trigger the airbag deployment and when ignited, this propellant produces nitrogen gas, which inflates the airbag. This process occurs very quickly—in less than one-twentieth of a second that is faster than the blink of an eye. Most air bags have internal tether straps that shape the fabric and limit the movement of the bag. Vents in the rear allow the bag to deflate slowly to cushion the head as it moves forward into the deploying air bag.[2]

Sensors deploy air bags only when deceleration exceeds a minimum threshold. If the change in speed due to an impact is lower than the threshold, the air bag will not inflate. In low to moderate speed collisions, the seat belt alone is usually sufficient to prevent serious injury. In high-speed crashes, the seat belt may not be able to prevent the driver's head from striking the steering wheel or the passenger's head from hitting the dashboard. Frontal air bags protect the head and upper body in frontal crashes.

A modern frontal airbag system consists of an electronic control unit (ECU) and one airbag module or two. The sensor (and micro-machined accelerometer) continuously monitors the acceleration and deceleration of the vehicle and sends this information to a microcontroller. When the microcontroller "recognizes" the crash pulse from the sensor, an electrical current is sent to the initiator in the airbag module.

1.4 Crash Sensors

Crash Sensors are the devices that work with the control module to discriminate between crash and non-crash events. These sensors measure the severity of the impact. They are set up so that sudden "negative acceleration" will cause the contacts to close, sending a signal to the control module which checks for a signal from the rear sensor which must arrive first to activate the airbag(s). It is important to note that at least 2 of these sensors must signal a crash before airbag deployment.

1.4.1 Sensors Types

By function, there are 2 types of sensors, Impact sensors and Safing sensors. The forward sensors are located in various locations forward of the passenger compartment. Some are located inside the fenders, some are on the cowl and some are attached to the core support in front of the radiator.

Rear sensors are also known as safing sensors as their function is to determine that a crash has occurred. Rear safing sensors are located in various locations in the passenger compartment depending on the manufacturer. Some are integrated with the Control/Diagnostic Module.

The rear safing sensor must close before the forward sensors to avoid airbag deployment in cases where the impact is not severe enough to cause deployment. When the vehicle is parked with the ignition off, deployment is very unlikely because there is no power to the circuits for deployment. This means that someone can hit the car and sound the alarm but not deploy the airbags.

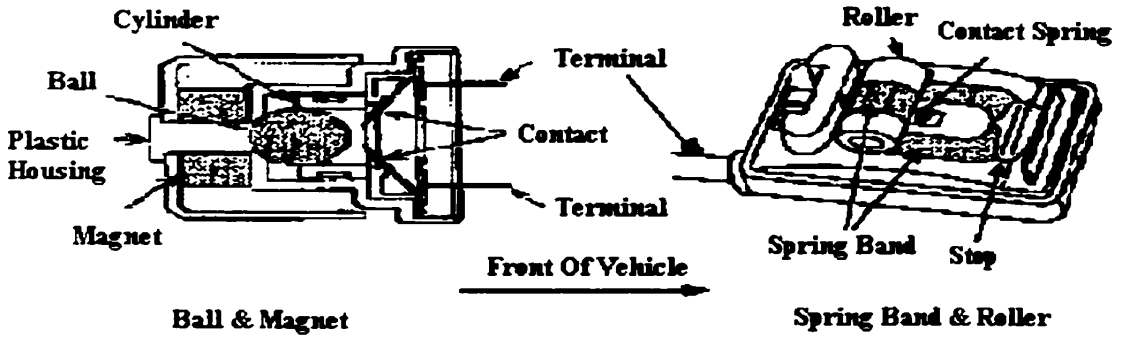

Fig:2 APeek Inside Forward Sensor

1.5 Inflator Assembly

When the Control Module activates the airbag assembly, an electric current is sent to the detonator, which ignites the Sodium Azide pellets. When it burns, it releases nitrogen gas very quickly and in large quantities. This is what inflates the airbag.

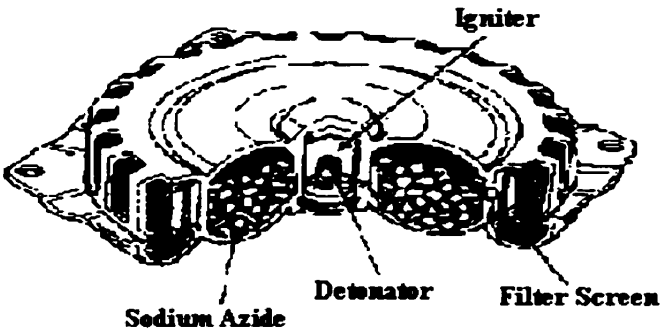

Fig:3 Typical Inflator Assembly Behind Streering Wheel

1.5.1 Sodium Azide

Sodium Azide is rocket fuel and is the fuel of choice for a number of reasons. It is a solid propellant with a very high gas generation ratio. It is very stable in this application. When Sodium azide burns, it's major product is Nitrogen gas, which makes up around 78% of the Earth's atmosphere. One of the other by-products is sodium hydroxide. This is commonly known as Lye, which is a caustic compound. The quantities produced are very small and present a very small risk of burns. The white powder residue seen after inflation is common corn starch, used as a lubricant for expansion of the airbag. Testing is underway with inflators that release argon gas.

1.6 Why Do Airbags Sometimes Cause Injuries?

In order to protect the head and upper body in high-impact crashes, air bags must inflate so quickly, and with such force, that they can cause injuries. While most of these injuries are minor, consisting only of bruises and abrasions, some are more serious, such as broken arms. In extreme cases, such as when the head or chest is against the module when it opens, fatal injuries can result.

People who sit close to the steering wheel are at higher risk of being injured by a deploying air bag than those who sit further away.

1.7 Future of Airbag Systems

It is now mandatory for air bags to be installed in all vehicles. In the case of air bags. In order to reduce the incidence of airbag induced injury.[3]

The problem of serious inflation injuries isn't going to be with us forever. Future airbag technologies will reduce the risk even among people who have moved forward before airbags inflate. Sensors will detect rear-facing restraints and automatically switch off passenger bags. Inflation rates will be tailored to crash severity. More advanced airbags could recognize people's positions just before inflating and reduce the force if someone is in position to be harmed.

Motor vehicle manufacturers are developing "smart" airbags. Already some possess two thresholds of activation, one that is appropriate for a belted occupant and another, lower threshold, for an unbelted person. The next generation of air bag systems will probably have proximity sensors that gauge how close an occupant is to the air bag module and will be equipped with warning systems that signal when someone is too close and automatic systems may prevent the airbag from deploying.

1.7.1 Smart Systems

The Smart Airbag System of the future is not just the airbag, but a redesign of the components in the current airbag systems. Features include:

- **Weight Sensors:** This is a new sensor for the passenger seat to classify the weight and to determine what type of occupant is in the seat, i.e.; adult or child.
- **Infrared Occupant Detection:** This system will use Infra-red beams (just like in a TV remote control) to detect the distance the passenger is from the airbag and adapt the force of deployment accordingly.
- **Capacitive Reflective Occupant Sensing:** These sensors will be located in the seat backs and in the dash to identify the distance of passengers from the

dashboard. These sensors will be able to discriminate between a human occupant and inanimate objects like groceries. This alone will save thousands of dollars in the cases where the driver is the only occupant in the front seat.

- **Updated Sensors:** The updated sensors will have the capabilities of deploying the seatbelt pre-tensioners faster, so in a crash situation passenger will be in the best position to benefit from the airbag deployment.
- **Centralized Electronic Control Unit:** The new control units will be able use all the input from the new sensor technology and through new software deploy what needed and when needed.[4]

1.8 Our Project Objective

To develop the technical bases for an airbag which will lead to the elimination of fatalities and reduce the severity of the injuries resulting from aggressive airbag deployment and optimizing the benefits to normally seated restrained occupants while also restoring the full protection for unbelted adults in high severity crashes.

To avoid the deployment of airbags when the passenger seat is empty. This approach will save the cost associated with recharging the airbag after unnecessary deployment.

To analyze the possibility of using fuzzy logic for this design approach and also try to overcome the RAM constraints in the microcontrollers.

To incorporate the information about occupant's weight and position into the airbag module to adjust the deployment speed of the airbag.

CHAPTER 2

History of Fuzzy Logic

Fuzzy logic theory was proposed by Professor Lotfi Zadeh of University of California, Berkeley in 1965.It was invented in the U.S. but engineered to perfection in Europe, and was mass-marketed in Japan. There are hundreds of successful fuzzy logic applications in today's world and have proved the value of this technology, but still some scientists condemn the concept.[6]

2.1 Industrial Applications

In 1970 Ebrahim Mamdani at Queen Mary College in London, England, used fuzzy logic to control a steam generator that he could not get under control with conventional techniques. Hans Zimmermann at the RWTH University of Aachen, Germany, used fuzzy logic for decision support systems. Other names given to fuzzy logic were "multi-valued logic" or "continuous logic" and fuzzy logic could not get broad acceptance in industry.[7]

Fuzzy logic gained more importance in decision support and data analysis applications in Europe around 1980 and advanced fuzzy logic technologies were developed in application and research projects. Most of these developments were modeled on human decision and evaluation process.[8]

2.2 Japanese leading role

Japanese companies started using fuzzy logic in control engineering around 1980 after inspiration from European fuzzy logic applications, the first. Fuzzy logic applications were at water treatment plant by Fuji Electric in 1983 and a subway system by Hitachi, which was opened in 1987. Most of the applications had dedicated fuzzy logic hardware because of poor computational performance of first fuzzy logic algorithms on standard hardware.[9]

Fuzzy logic supports the generation of a fast prototype and incremental optimization and fuzzy logic system always remains plain and simple to understand. The "intelligence" of a system is not buried in differential equations or source code.

As a result fuzzy logic is now used in about every application area for intelligent control or data processing. Photo and video cameras use fuzzy logic to put photographer's expertise in their control. Mitsubishi announced the world's first car where every control system is based on fuzzy logic and most other Japanese car manufacturers use fuzzy logic in some of their components. In factory automation, Omron Corp. Claims more than 350 patents and fuzzy logic control optimizes many chemical and biological processes. [20], [28]

2.3 Europe Chases Japan

When European corporations realized that they have almost lost a key technology to Japan, they started a major effort to promote fuzzy logic in their applications. Now a lot

of successful fuzzy logic mass market products have been launched in Europe and uncounted number of industrial automation and process control applications are successfully using fuzzy logic. The fuzzy logic enhanced products include home appliances that realized mayor savings in energy and water consumption with no added product costs as well as many automotive applications. The industrial automation applications include chemical and biological process control, machinery equipment control, and intelligent sensors.[27]

Due to the big commercial success of these applications, fuzzy logic is now considered a "standard" design technique and has gained broad acceptance in the engineering community. One of the supporting factors was the advent of advanced fuzzy logic software design tools that supported all development stages of a fuzzy logic design. Some of the fuzzy logic design tool software houses teamed up with major semiconductor and industrial automation equipment manufacturers to provide seamless development environment for most target hardware platforms.

2.4 Fuzzy Logic in the North America

Fuzzy logic has recently gained a lot of interest among companies who are in heavy competition with both Asia and Europe. There are many factors that count. North American manufacturers do not compete with the Japanese in entertainment electronics manufacturing. Use of fuzzy logic in camcorders, cameras, and hi-fi is more a competitive factor between Japanese corporations themselves. In Europe, most

applications are in industrial automation and automotive engineering. In other applications, there is tough competition from both Europe and Japan.[28]

Fuzzy logic proved to be an excellent tool to build decision support systems, memory cache and hard disk controllers as well as compression algorithms for speech and video. Also, telecom applications such as echo cancellation, network routing, and speech recognition benefit from fuzzy logic. All fuzzy logic experts agree that the clever combination of neural network technologies and fuzzy logic will be the next logical step in further developing the technology. [6]

CHAPTER 3

BASIC CONCEPTS OF FUZZY LOGIC

3.1 Fuzzy Expert Systems

In the real world there exists much fuzzy knowledge, i.e., knowledge that is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature. Human thinking and reasoning frequently involve fuzzy information, possibly originating from inherently inexact human concepts and matching of similar rather then identical experiences. In systems based upon classical set theory and two-valued logic, it is very difficult to answer some questions because they do not have completely true answers. Humans, however, can give satisfactory answers, which are probably true. Expert systems should not only give such answers but also describe their reality level. This level should be calculated using imprecision and the uncertainty of facts and rules that were applied. Expert systems should also be able to cope with unreliable and incomplete information and with different expert opinions. Fuzziness and uncertainty are the two distinct inexact concepts employed in the system. The following sections will discuss the general theory of both fuzziness and uncertainty, their implications on rule evaluation and algorithms implemented for extracting exact values from fuzzy facts.[19]

3.2 Fuzziness

Fuzziness occurs when the boundary of a piece of information is not clear-cut. For example, concepts such as *young*, *tall*, *good*, or *high* are fuzzy. There is no single quantitative value which defines the term *young*. For some people, age 25 is *young*, and for others, age 35 is *young*. In fact the concept *young* has no clean boundary. Age 1 is definitely *young* and age 100 is definitely not *young*; however, age 35 has some possibility of being *young* and usually depends on the context in which it is being considered. The representation of this kind of information is based on the concept of fuzzy set theory [10]. Unlike classical set theory where one deals with objects whose membership to a set can be clearly described, in fuzzy set theory membership of an element to a set can be partial, i.e., an element belongs to a set with a certain grade (possibility) of membership. More formally a fuzzy set A in a universe of discourse U is characterized by a membership function

$$\mu_A : U \rightarrow [0,1] \tag{1}$$

which associates with each element x of U a number $\mu_A(x)$ in the interval $[0,1]$ which represents the grade of membership of x in the fuzzy set A.

For example, the fuzzy term *young* might be defined by the fuzzy set in Table below.

Age	Grade of Membership
25	1.0
30	0.8
35	0.6
40	0.4
45	0.2
50	0.0

Fuzzy Term *young*

Regarding equation (1), one can write

$$\mu_{young}(25) = 1,\ \mu_{young}(30) = 0.8,\ \ldots\ ,\ \mu_{young}(50) = 0$$

Grade of membership values constitute a possibility distribution of the term *young*. The table can also be shown graphically

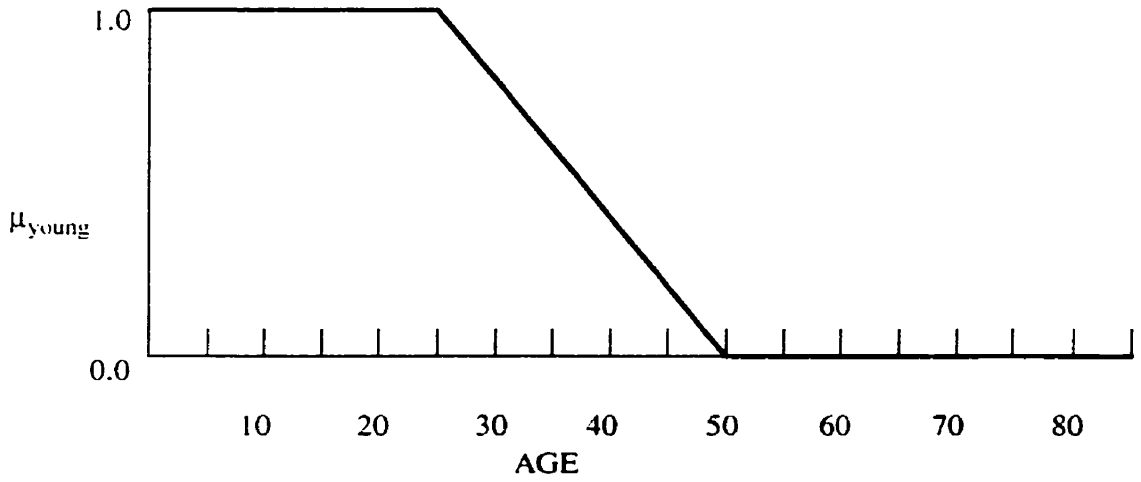

Fig: 4 Possibility distribution of *young*

3.2.1 Uncertainty

Uncertainty occurs when one is not absolutely certain about a piece of information. The degree of uncertainty is usually represented by a crisp numerical value on a scale from 0 to 1, where a certainty factor 1 of 1 indicates that the expert system is very certain that a fact is true, and a certainty factor of 0 indicates that it is very uncertain that a fact is true.

A fact is composed of two parts: the fact in the sense of standard and its certainty factor. In general a fact takes the following form:

(fact) [CF certainty factor]

The CF acts as the delimiter between the fact and the certainty factor and [ ] indicates an optional part. For example, (prediction sunny) CF 0.8 is a fact that indicates that the weather will be sunny with a certainty of 80%. However, if the certainty factor is omitted, (prediction sunny) then we assume that the weather will be sunny with a certainty of 100%.

3.3 Inference Techniques

Rule evaluation depends on a number of different factors, such as whether or not fuzzy variables are found in the antecedent or consequent part of a rule, whether a rule contains multiple antecedents or consequents, whether a fuzzy fact being asserted has the same fuzzy variable as an already existing fuzzy fact (global contribution), and so on.

3.3.1 Simple Rules

Consider the simple rule of form

if A then C	CF_r
A'	CF_f

C'	CF_c

where:

A is the antecedent of the rule

A' is the matching fact in the fact database

C is the consequent of the rule

C' is the actual consequent calculated

CF_r is the certainty factor of the rule

CF_f is the certainty factor of the fact

CF_c is the certainty factor of the conclusion

Three types of simple rules are defined: CRISP_, FUZZY_CRISP, and FUZZY_FUZZY. If the antecedent of the rule does not contain a fuzzy object, then the type of rule is CRISP_ regardless of whether or not a consequent contains a fuzzy fact. If only the

antecedent contains a fuzzy fact, then the type of rule is FUZZY_CRISP. If both antecedent and consequent contain fuzzy facts, then the type of rule is FUZZY_FUZZY.

3.3.2.1 CRISP_ Simple Rule

If the type of rule is CRISP_, then A' must be equal to A in order for this rule to fire. This is a "normal" rule (actually A would be a pattern and A' would match the pattern specification, but for simplicity we will not deal with patterns). In that case the conclusion C' is equal to C, and

$$CF_c = CF_r * CF_f \tag{2}$$

3.3.1.2 FUZZY_CRISP Simple Rule

If the type of rule is FUZZY_CRISP, then $\acute{A}$ must be a fuzzy fact 1 with the same fuzzy variable as specified in A for a match to occur and the rule to be placed on the agenda. In addition, while values of the fuzzy variables A and $\acute{A}$ represented by the fuzzy sets F_α and F'_α do not have to be equal, they must overlap. For example, the fuzzy facts (*temperature high*) and (*pressure high*) do not match because the fuzzy variables *temperature* and *pressure* are not the same. However, given the fuzzy facts (*pressure low*), (*pressure medium*), and (*pressure high*), as illustrated in Figure, clearly (*pressure low*) and (*pressure medium*) overlap and thus match, while (*pressure low*) and (*pressure high*) do not match.

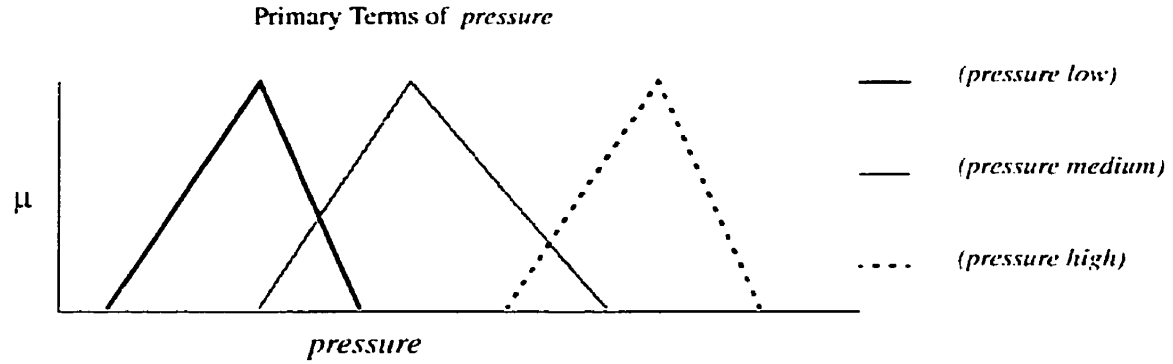

Fig: 5 Matching of fuzzy facts

For a FUZZY_CRISP rule, the conclusion C' is equal to C, and

$$CF_c = CF_r * CF_f * S$$

where S is a measure of similarity between the fuzzy sets F_α (determined by the fuzzy pattern A) and F'_α (of the matching fact A'). The measure of similarity is based upon the measure of possibility P and the measure of necessity N. It is calculated according to the following formula [11], [32], [34]

$$S = P(F_\alpha / F'_\alpha) \qquad \text{if } N(F_\alpha / F'_\alpha) > 0.5$$

$$S = (N(F_\alpha / F'_\alpha) + 0.5) * P(F_\alpha / F'_\alpha) \qquad \text{otherwise}$$

where

$$P(F_\alpha / F'_\alpha) = \max(\min(\mu F_\alpha(u), \mu F'_\alpha(u))), \qquad \forall u \in U$$

And

$$N(F_\alpha / F'_\alpha) = 1 - (F_\alpha / F'_\alpha)$$

F'_α is the complement of F_α described by the following membership function

$$\mu F'_\alpha(u) = 1 - \mu F_\alpha(u) \qquad \forall u \in U$$

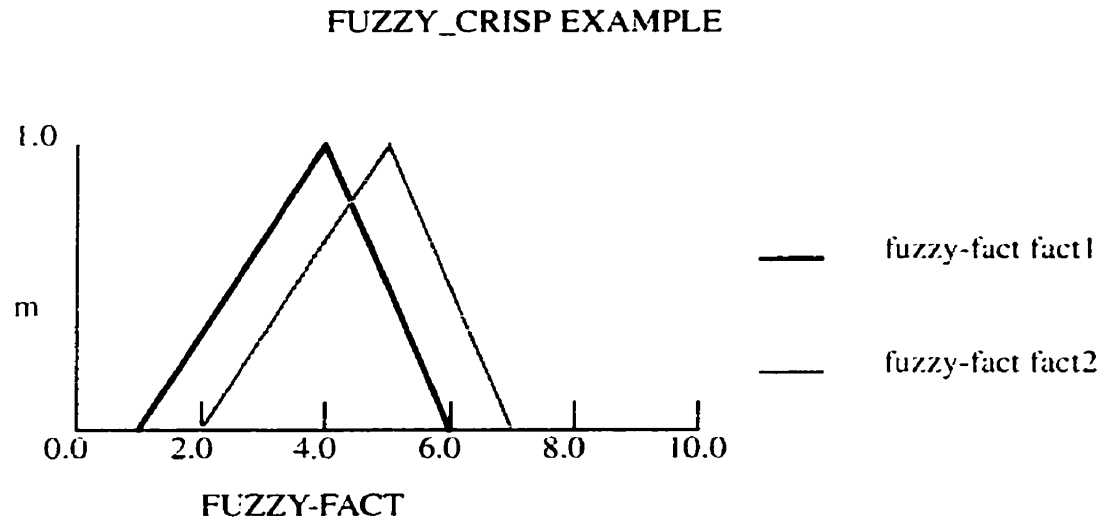

Fig: 6 Fact and antecedent fuzzy sets

Therefore, if the similarity between the fuzzy sets associated with the fuzzy pattern (A) and the matching fact (A') is high the certainty factor of the conclusion is very close to $CF_r * CF_f$ since S will be close to 1. If the fuzzy sets are identical then S will be 1 and the certainty factor of the conclusion will equal $CF_r * CF_f$. If the match is poor then

this is reflected in a lower certainty factor for the conclusion. Note also that if the fuzzy sets do not overlap then the similarity measure would be zero and the certainty factor of the conclusion would be zero as well. In this case the conclusion should not be asserted and the match would be considered to have failed and the rule would not be placed on the agenda.

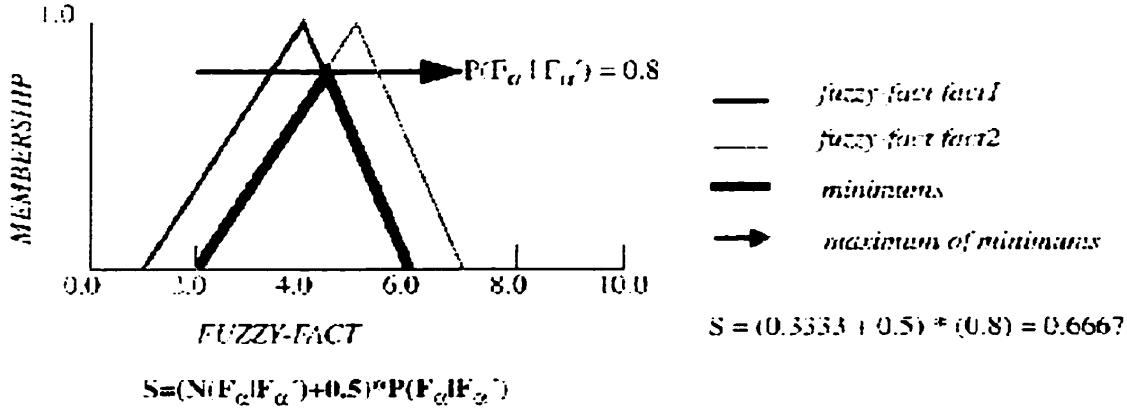

Fig:7 Similarity Calculation

First, the necessity is calculated as in Figure below.

Since the necessity is less than 0.5,

$$S = (N (F_\alpha / F'_\alpha) + 0.5) * P(F_\alpha / F'_\alpha)$$ (see Figure below)

And thus $CF_c = (0.7) * (0.8) * (0.6667) = 0.3733.$

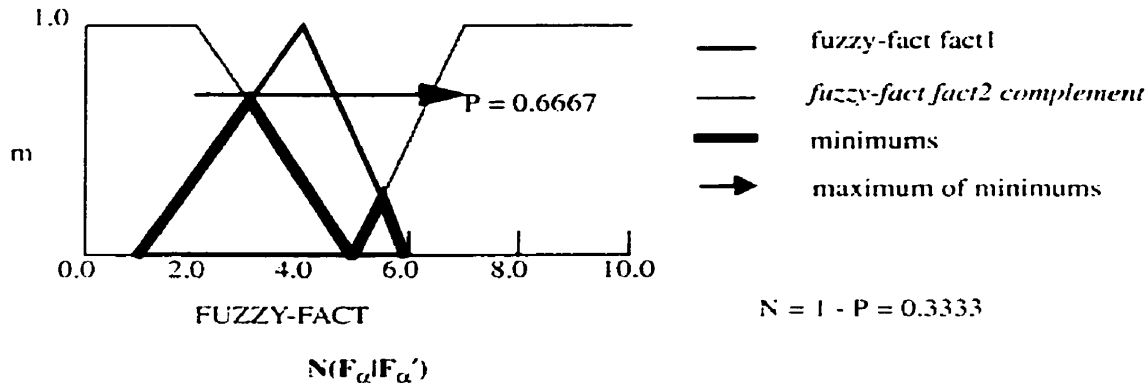

Fig: 8 NECESSITY CALCULATION

3.3.1.3 FUZZY_FUZZY Simple Rule

If the type of rule is FUZZY_FUZZY, and the fuzzy fact and antecedent fuzzy pattern match in the same manner as discussed for a FUZZY_CRISP rule, then it is shown in [16] that the antecedent and consequent of such a rule

are connected by the fuzzy relation

$$R = F_{\alpha} * F_{c}$$

Where

F_{α} is a fuzzy set denoting the value of the fuzzy antecedent pattern

F_{c} is a fuzzy set denoting the value of the fuzzy consequent.

In the current version of the system the membership function of the relation R is calculated according to the formula.

$$\mu_R(u, v) = min(\mu_{F'_\alpha}(u), \mu_{F_c}(v)), \quad \forall(u, v) \in U \times V$$

Other algorithms for forming this relation can be found in [12]. The calculation of the conclusion is based upon the compositional rule of inference [13], which can be described as follows:

$$\boldsymbol{F'_c = F'_\alpha \circ R}$$

where F'_c is a fuzzy set denoting the value of the fuzzy object of the consequent. The membership function of F'_c is calculated as follows [14], [37]

$$\mu_{F'_c}(v) = max_{u \in U}(min(\mu_{F'_\alpha}(u), \mu_R(u, v)))$$

which may be simplified to

$$\mu_{F'_c}(v) = min(z, \mu_{F_c}(v))$$

where

$$z = max\left(min\left(\mu_{F'_\alpha}(u), \mu_{F_\alpha}(u)\right)\right)$$

The certainty factor of the conclusion is calculated according to

$$CF_c = CF_r * CF_f$$

A graphical illustration of the matching of the fuzzy fact with the fuzzy pattern and the generation of the fuzzy conclusion is shown below in Figure 8. Note that this type of inference method is commonly referred to as max-min rule of inference. The conclusion set is simply *clipped* off at the z value. Figure shows the same results using a max-prod rule of inference. In this case the conclusion has all of its membership values scaled by the z value.

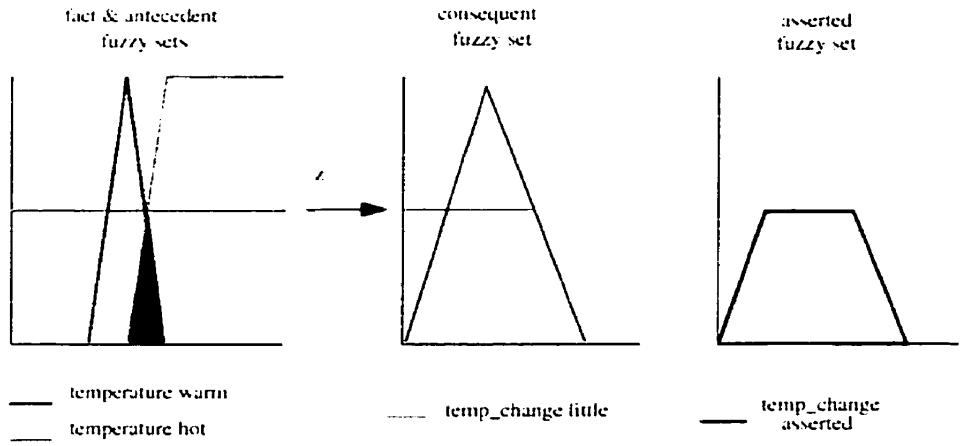

Fig: 9 Compositional rule of inference (max-min)

3.3.4. Complex Rules

3.3.4.1 Multiple Consequents

The consequent part of the rule can only contain multiple patterns (C_1 , C_2 ,... C_n) with the implicit and conjunction between them. They are treated as multiple rules with a single consequent. So the following rule:

if Antecedents then C_1 and C_2 and ... and C_n

is equivalent to the following rules:

if Antecedents then C_1

if Antecedents then C_2

...

if Antecedents then C_n

3.3.4.2 Multiple Antecedents

From the above, clearly, only the problem of multiple patterns in the antecedent with a single assertion in the consequent needs to be considered. If the consequent assertion is not a fuzzy fact, no special treatment is needed since the conclusion will be the crisp (non-fuzzy) fact. However, if the consequent assertion is a fuzzy fact, the fuzzy value is calculated using the following basic algorithm [15], [37].

If logical *and* is used, one has

if A_1 *and* A_2 then C	CF_r
A'_1	CF_{f1}
A'_2	CF_{f2}

C'	CF_c

where A'_1 and A'_2 are facts (crisp or fuzzy) that match the antecedents A_1 A_2 respectively. In this case the fuzzy set describing the value of the fuzzy assertion in the conclusion is calculated according to the formula

$$F'_c = F'_{c1} \cap F'_{c2}$$

where

$\cap$ denotes the intersection of two fuzzy sets

F'_{c1} is the result of fuzzy inference for the fact A'_1 and the simple rule

if A_1 then C

F'_{c2} is the result of fuzzy inference for the fact A'_2 and the simple rule

if A_2 then C

In next figure we see the results of a rule in which both A_1 and A_2 are fuzzy patterns. Note that if both A_1 and A_2 were crisp (non-fuzzy) facts then the conclusion would just be the fuzzy fact C since we would be dealing with two CRISP_FUZZY simple rules. If one of the patterns is crisp (say A_1) and the other is fuzzy then the conclusion is F'_{c2} since the CRISP_FUZZY simple rule would conclude C and the FUZZY_FUZZY simple rule would conclude F'_{c2}. The intersection of these two would just be F'_{c2}.

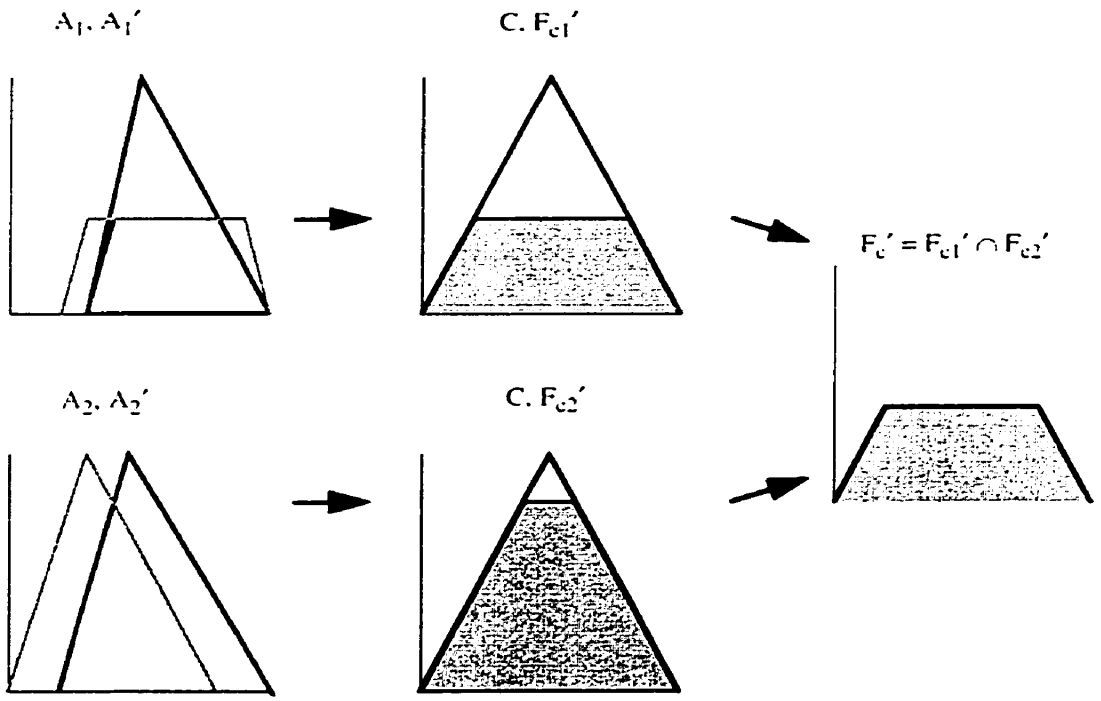

Fig: 10 Compositional rule for multiple antecedents

The certainty factor of the conclusion is calculated according to MYCIN's model

$$CF_c = min(CF'_{f1}, CF'_{f2}) * CF_r$$

where *min* denotes the minimum of the two numbers and CF'_{f1} is the *CF* of the simple rule if A_1 then *C* given the matching fact A'_1. CF'_{f2} is the *CF* of the simple rule if A_2 then *C* given the matching fact A'_2

The above algorithm can be applied repeatedly to handle any combination of antecedent patterns, i.e.,

$$F'_c = F'_{c1} \cap F'_{c2} \ldots \cap F'_{cn}$$

$$CF'_c = min(CF'_{f1}, CF'_{f2}, \ldots, CF'_{fn}) * CF_r$$

3.4 Defuzzification

The outcome of the fuzzy inference process is a fuzzy set, specifying a fuzzy distribution of a conclusion. However, in some cases, such as control applications, only a single discrete action may be applied, so a single point that reflects the best value of the set needs to be selected. This process of reducing a fuzzy set to a single point is known as *defuzzification*.

There are several possible methods, each one of which has advantages and disadvantages. A method which has been widely adopted is to take the center of gravity (COG or moment) of the whole set. This has the advantage of producing smoothly varying

controller output, but it is sometimes criticized as giving insufficient weight to rule consequents that agree and ought to reinforce each other. Another method concentrates on the values where the possibility distribution reaches a maximum, called the mean of *maxima* method. The mean of *maxima* (*MOM*) algorithm is criticized as producing less smooth controller output, but has the advantage of greater speed due to fewer floating point calculations.[6], [38], [39]

3.4.1 Center of Gravity Algorithm

The center of gravity method may be written formally as

$$x^* = \frac{\int_{(x \in U)} (x \cdot f(x))dx}{\int_{(x \in U)} f(x)dx}$$

where x^* is the recommended, defuzzified value, and the universe of discourse is U. The integral then reduces to a simple summation, where x_i' is the local center of gravity, A_i is

$$x^* = \frac{\sum_{i=1}^{n} x_i' \cdot A_i}{\sum_{i=1} A_i}$$

the local area of the shape underneath line segment (p_{i-1} , p_i), and n is the total number of points.

For each shaded subsection in above figure, the area and center of gravity is calculated according to the shape identified (i.e., triangle, rectangle or trapezoid). The center of gravity of the whole set is then determined:

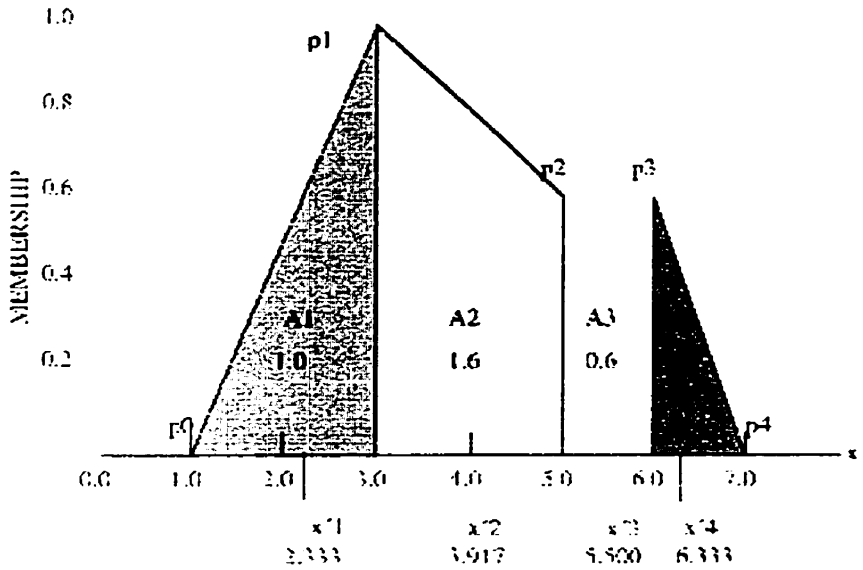

Fig: 11 **Example of COG defuzzification**

$$x' = \frac{2.333 \cdot 1.0 + 3.917 \cdot 1.6 + 5.5 \cdot 0.6 + 6.333 \cdot 0.3}{1.0 + 1.6 + 0.6 + 0.3} = 3.943$$

3.4.2 Mean of Maxima Algorithm:

The MOM algorithm returns the x-coordinate (x") of the point at which the maximum membership (y) value of the set is reached. If the maximum y value is reached at more than one point, then the average of all the x" is taken.[16], [6]

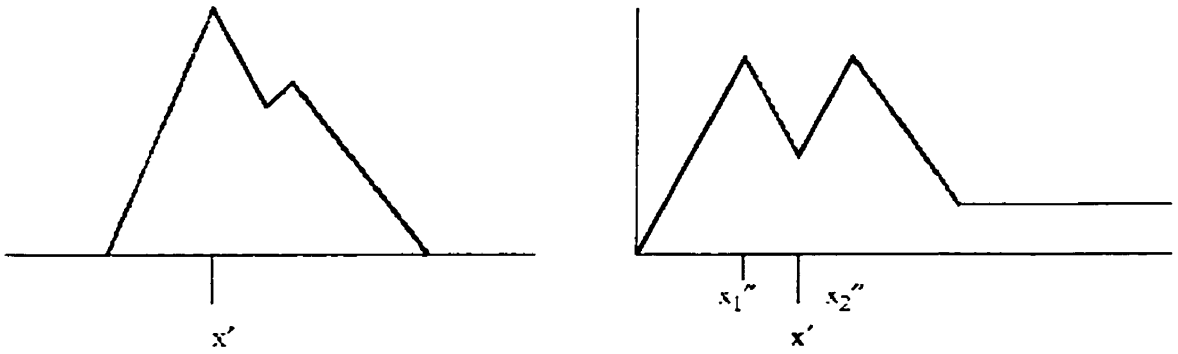

Fig: 12 **Examples of MOM defuzzification**

3.4.3 Center of Maximum

we derive the centroid equation for the sum rule which illuminates the assumptions made in deriving the defuzzification.

Let $L_i(y)$ be the original output membership function associated with rule i, where y is the output universe of discourse. After applying rule i, this membership function will be reduced to the value[16]

$$m_i(y) = w_i L_i(y)$$

Where w_I is the minimum weight found by applying rule i. The sum of these reduced output membership functions over all rules is given by

$$M(y) = \sum_{i=1}^{N} m_i(y)$$

Where N is the number of rules.

$$y_0 = \frac{\int yM(y)dy}{\int M(y)dy}$$

The crisp output value y_0 is then given by the centroid of $M(y)$ from the equation

$$I_i = \int L_i(y)dy$$

Note that the centroid of membership function $L_i(y)$ is given by

$$c_i = \frac{\int y L_i(y) dy}{\int L_i(y) dy}$$

But

$$I_i = \int L_i(y) dy$$

is just the area of membership function $L_i(y)$. Substituting the values, we get

$$\int y L_i(y) dy = c_i I_i$$

So we can write the numerator as

$$\int y M(y) dy = \int y \sum_{i=0}^{N} w_i L_i(y) dy$$

$$\int y M(y) dy = \sum_{i=1}^{N} \int y w_i L_i(y) dy$$

$$\int y M(y) dy = \sum_{i=1}^{N} w_i c_i I_i$$

Similarly the denominator can be written as

$$\int M(y)dy = \int \sum_{i=0}^{N} w_i L_i(y)dy$$

$$\int M(y)dy = \sum_{i=1}^{N} \int w_i L_i(y)dy$$

$$\int M(y)dy = \sum_{i=1}^{N} w_i I_i$$

So we can write the crisp output of fuzzy controller as

$$y_0 = \frac{\sum_{i=1}^{N} w_i c_i I_i}{\sum_{i=1}^{N} w_i I_i}$$

The above equation says that we can compute the output centroid from the centroids, c_I, of the individual output membership function.

We also note that the summation is over all N rules. But the number of output membership function ,Q, will in general, be less than the number of rules ,N. That means that there will be many terms that have the same value of c_I and I_i. For example, suppose that rules 2, 3, and 4 in the sum all have the output membership function L^k as the consequent. This means that in the sum,

$$w_2 c_2 I_2 + w_3 c_3 I_3 + w_4 c_4 I_4$$

the values c_l and I_l are the same values c^k and I^k because they are just the centroid and area of the *kth* output membership function. These three terms would contribute the value

$$(w_2 + w_3 + w_4)c^k I^k = W^k c^k I^k$$

to the sum where

$$(w_2 + w_3 + w_4) = W^k$$

is the sum of all weights from rules whose consequent is output membership function L^k.

This means that the equation for the output value, y_0, can be written as

$$y_0 = \frac{\sum_{k=1}^{Q} W^k c^k I^k}{\sum_{k=1}^{Q} W^k I^k}$$

If the area of all output membership function I^k are equal then the equation reduces to

$$y_0 = \frac{\sum_{k=1}^{Q} W^k c^k}{\sum_{k=1}^{Q} W^k}$$

Equations above show that the output crisp value of a fuzzy controller can be computed by summing over only the number of output membership functions rather than over all fuzzy rules. Also, if we want to compute the output crisp value, then we need to specify only the centroids, C^k, of the output fuzzy membership functions. This is equivalent to assuming singleton fuzzy sets for the output.

We use singleton fuzzy sets for the output represented by the centroids, C^k. We also use the MIN-MAX inference rule It should be clear that in this case the centroid Y_o will still be given where W^k is now the output array

CHAPTER 4

68HC12 FEATURES

4.1 Introduction

The 68HC12 is a high-speed, 16-bit processing unit that has a programming model identical to that of the industry standard M68HC11 CPU. The 68HC12 instruction set is a proper superset of the M68HC11 instruction set, and M68HC11 source code is accepted by 68HC12 assemblers with no changes.

The 68HC12 has full 16-bit data paths and can perform arithmetic operations up to 20 bits wide for high-speed math execution.5

Unlike many other 16-bit CPUs, the 68HC12 allows instructions with odd byte counts, including many single-byte instructions. This allows much more efficient use of ROM space.

An instruction queue buffers program information so the CPU has immediate access to at least three bytes of machine code at the start of every instruction.

In addition to the addressing modes found in other Motorola MCUs, the 68HC12 offers an extensive set of indexed addressing capabilities including:

- Stack pointer can be used as an index register in all indexed operations
- Program counter can be used as an index register in all but auto inc/dec mode
- Accumulator offsets allowed using A, B, or D accumulators

• Automatic pre- or post-, increment or decrement (by –8 to +8)

• 5-bit, 9-bit, or 16-bit signed constant offsets

•16-bit offset indexed-indirect and accumulator D offset indexed-indirect addressing.

4.2 Memory Structure

The MC68HC912B32 has a large 32k byte flash EEPROM for program memory, 1k bytes of static RAM, and 768 bytes of byte-erasable EEPROM. This is the first MCU to include both flash EEPROM and byte-erasable EEPROM on the same chip. An external 12 volt supply is used to erase and program the flash memory, the byte-erasable EEPROM uses only the normal 2.7 to 5.5 volt supply for programming and erase operations. The MC68HC912B32 can be used in expanded mode systems but its multiplexed 16-bit address/data bus is primarily intended for factory testing.

MC68HC812A4	MC68HC912B32
16-bit CPU	16-bit CPU
2.7 - 5.5 Volt Operation	2.7 - 5.5 Volt Operation
Programmable PLL or Xtal = 2x Bus Rate	Xtal = 2x Bus Rate
112 pins (up to 95 General Purpose I/O) Key Wakeup on Three 8-Bit Ports	80 pins (up to 64 General Purpose I/O)
Expanded Non-mux Bus or Single Chip 16-Bit Wide or 8-Bit Narrow	Single Chip or Expanded Multiplexed Bus 16/16 Wide or 16/8 Narrow
Memory Expansion to >5 megabytes	
6 Programmable Chip Selects	
4K EEPROM 1K SRAM 256 Byte Register Space	32K Flash EEPROM 768 EEPROM 1K SRAM 256 Byte Register Space
8 Channel, 8-Bit A/D	8 Channel, 8-Bit A/D
8 Channel Timer	8 Channel Timer
16-Bit Pulse Accumulator	16-Bit Pulse Accumulator
	4 Channel PWM
2 Channels Asynchronous SCI	1 Channel Asynchronous SCI
1 Channel Synchronous SPI	1 Channel Synchronous SPI
	1 Channel J1850 Digital Serial
Single-Wire BDM	Single-Wire BDM with Hardware Breakpoints
COP Watchdog Timer and Clock Monitor	COP Watchdog Timer and Clock Monitor
Periodic Interrupt Timer	Periodic Interrupt Timer

4.3 Arithmetic Logic Unit

68HC12 has a 16-bit processor with an ALU that is as wide as 20 bits for some operations. All data busses in the M68HC12 are 16-bits and the external bus interface is normally 16 bits. An instruction queue (similar to a pipeline) that caches program information so that at least two more bytes of object code, in addition to the 8-bit op-code, are visible to the CPU at the start of execution for all instructions so many instructions can execute in a single cycle with no delays for fetching additional program information. Program information is fetched into the instruction queue 16 bits at a time but instructions can be any length from one byte to six bytes. This allows 68HC12 object code to be more efficient. In the 68HC12, all indexed instructions have an op-code, a post-byte, and 0, 1, or 2 extension bytes to specify index offsets. The post-byte code specifies which index register to use as the base reference and the type of indexed addressing. It allows X, Y, SP, or PC to be used as the base index register and has seven types of indexed addressing.

For offsets of -16 through +15, the 5-bit offset is included in the post-byte and has a signed 5-bit offset mode, a signed 9-bit offset mode, and a 16-bit offset mode so an instruction like LDAA 4,X would be encoded into two bytes of object code. The accumulator offset indexed mode allows 8-bit accumulators A or B or the 16-bit D accumulator to be used as an additional offset that is added to the base index register to form the effective address of the instruction's operand.

The 68HC12 has a new form of auto- pre/post increment/decrement by -8 through +8. For example, LDA ,X++ would adjust the index by one because the operand was a byte

while LDX ,Y++ adjusted the index by two because the operand was a word. In the 68HC12, we can say LDAA 5,SP- which loads a byte into accumulator A and then post-decrements SP by 5. This flexibility allows the 68HC12 to work very efficiently with small data structures. Execution time for these instructions is the same as it would be for the simple no-offset case (LDAA 5,+SP takes only three bus cycles). This sub-mode of indexed addressing allows X, Y, or SP to be used as the base index register, but not PC because that would interfere with the normal sequence of execution of instructions. These instructions are especially useful with move instructions such as MOVW 2,SP+,2,X+ which pulls a word from the stack and stores it at 0,X and automatically post-increments SP and X by 2 each.

4.4 Addressing Modes

68HC12 has two types of indexed-indirect indexing, D accumulator offset, and 16-bit offset. These instructions use X, Y, SP, or PC as the base index register, form an intermediate address by adding D or a 16-bit offset, fetch the 16-bit value from that address, and finally use this fetched value as the effective address to access the operand of the original instruction. This type of indexing can be used to program computed GOTO type constructs, to access operands in a position independent way, or to program some types of case or switch statements in C.

The enhanced indexed addressing is one of the strongest features of the 68HC12 for all kinds of programmers, but the SP relative indexing is especially important for C-compilers. The 68HC12 also added LEAX, LEAY, and LEAS instructions which provide an easy way to do pointer arithmetic. For example a 5-, 9-, or 16-bit signed constant can

be added to (or subtracted from) the index, A, B, or D can be added to the index (D may be thought-of as a signed or unsigned value since it is the same width as the address bus), or the index can be replaced by a value from memory (using indexed indirect modes).

4.4.1 Indexed Addressing Modes.

The 68HC12 has a general transfer/exchange instruction that uses a post-byte to choose transfer or exchange and to specify which registers are involved. Some of these combinations that involve transfer or exchange of an 8-bit register to a 16-bit register, perform sign extension or zero extension as an added bonus.

It has bit manipulation instructions (BSET, BCLR, BRSET, and BRCLR) to make the instruction set more. Register stacking instructions include instructions for pushing and pulling CCR and D registers.

Since this is a full 16-bit CPU, so 68HC12 can do 8x8 or 16x16 multiply operations in three bus cycles. Divide operations take 11 or 12 cycles depending upon which divide instruction it is. There are also some specialty math instructions including a 16x16 to 32-bit signed multiply-and-accumulate instruction and table lookup-and-interpolate instructions for tables of 8- or 16-bit entries.

4.5 Instruction Set

Possibly the most unusual instructions in the 68HC12 are its four fuzzy logic instructions which do membership function calculations, rule evaluation with weighted or unweighted rules, and a fourth fuzzy logic instruction that calculates the sum-of-products and sum-of-weights needed to do weighted average defuzzification. These fuzzy instructions allow a

complete fuzzy inference kernel to be programmed in about 50 bytes and execute in about 60 microseconds. This is better than a 5:1 improvement in program size and better than 10x improvement in speed when compared against a 4 MHz bus rate MC68HC11.

68HC12 improvements continue with 8- and 16-bit memory-to-memory moves that work with all practical combinations of immediate, extended, and indexed addressing modes. These instructions are especially useful in a register-based architecture and allow data movement without using up any CPU registers. The 68HC12 also has a complete set of long branches for signed and unsigned conditional branching. And there is a new group of loop primitive instructions (DBEQ, DBNE, IBEQ, IBNE, TBEQ, and TBNE) which use A, B, D, X, Y, or SP as the loop counter. The loop counter is decremented, incremented, or tested and then a branch is taken on the condition that the counter has reached zero or has not reached zero.

Another group of instructions for doing MIN or MAX operations between an accumulator (A or D) and a byte or word sized memory location. There are versions of each that place (overwrite) the result into either the accumulator or the memory location. This results in a total of eight instructions in this group (MINA, MINM, MAXA, MAXM, EMIND, EMINM, EMAXD, and EMAXM).

4.6 Small Comprehensive Control Interfac

In development systems, smaller is better when it comes to the physical and resource requirements to gain access to a target application system. In many products it is unreasonable to connect a large umbilical connector that clamps over the main processor. The desire to debug systems in their final packaged form has led to various schemes that

rely on serial communication to minimize the number of connections to the target system. Some of these systems also rely on some sort of background task that runs within the target system to respond to requests and commands that were passed into the system through a serial port. Typically the actual operation of debug commands and the real time operation of the target application are mutually exclusive.

The M68HC12 takes this technology to a new level in both the reduction of the physical interface and greater separation of debug and target application functions. The physical interface for the background system in the 68HC12 is a single MCU pin that does not share functions with any target application functions. Background access can be gained in any 68HC12 target application system by connecting a common ground and this single communication wire. .[5], [40], [42]

With this trivial connection, the host can read or write any location in the 64 kbyte map of the target MCU without stopping or slowing down the real time operations of the target application. For other debug functions such as reading and writing CPU registers, tracing single instructions in the application code, reading and writing whole blocks of memory, and accessing other development features, the target application can be stopped (forced to an active background debug mode) to wait for serial commands.

Products based on this MCU can be fully assembled before the on-chip flash memory is programmed with target application code. The background debug interface can be used to program or reprogram flash or byte-erasable EEPROM after final assembly. This interface can also be used for maintenance modifications to application code or for product troubleshooting in the field.

CHAPTER 5

DESIGN CONCEPTS

5.1 Stochastic Uncertainty

Stochastic uncertainty deals with the uncertainty toward the occurrence of a certain event. Consider the statement:

"The probability of hitting the target is 0.8'

The event itself -- hitting the target -- is well defined. The uncertainty in this statement is whether the target is hit or not. This uncertainty is quantified by a degree of probability. In the case of the statement, the probability is 0.8. Statements like this can be processed and combined with other statements using stochastic methods.[6]

5.2 Lexical Uncertainty

Uncertainty also lies in human languages, the so-called lexical uncertainty. It deals with the imprecision that is inherent to most words humans use to evaluate concepts and derive conclusions. Consider words such as "tall men", "hot days", or "stable currencies", where no exact definitions underlie. Whether a man is considered "tall" hinges on many factors. A child has a different concept of a "tall" man than an adult. Also the context and the background of a person making an evaluation plays a role. Even for one single person, an exact definition on whether a man is considered "tall" does not exist. No law exists determines the threshold above which a man is conceived "tall". This would not make sense anyhow, since a law that defines all men taller than 6' 4" to be "tall" would

imply that a man with 6' 3" is not tall at all. The science that deals with the way humans evaluate concepts and derive decisions is psycholinguistics. It has been proven that humans use words as "subjective category" to classify figures such as "height", "temperature", and "inflation". Using these subjective categories, things in real world are evaluated to which degree they satisfy the criteria.

Even though most concepts used are not precisely defined, humans can use them for quite complex evaluations and decisions that are based on many different factors. By using abstraction and by thinking in analogies, a few sentences can describe complex contexts that would be very hard to model with mathematical precision. Consider the statement:

"We will probably have a successful financial year"

On a first glance, second example is very similar than first. However, there are significant differences. First, the event itself is not clearly defined. For some companies, a successful financial year means that they deferred bankruptcy, for others it means to have surpassed last years profit. Even for one company, no fixed threshold exists to define whether a fiscal year is considered to be successful or not. Hence, the concept of a "successful fiscal year" is a subjective category.

Another difference lies in the definition of expressing probability. While in statement 1, the probability is expressed in a mathematical sense, second example does not quantify a probability. If someone expresses that a certain type of airplane probably has problems, the actual probability can well be lower than 10%, still justifying this judgement. If someone expresses that the food in a certain expensive restaurant is probably good, the

actual probability can well be higher than 90%. Hence, the expression of probability in statement 2 is a perceived probability rather than a mathematically defined probability as in statement 1. In statement 2, the expression of probability is also a subjective category just as "tall men".

5.3 Modeling Linguistic Uncertainty

Statements using subjective categories such as second example play a major role in the decision making process of humans. Even though these statements do not have quantitative contents. humans can use them successfully for complex evaluations. In many cases the uncertainty that lies in the definition of the words we use, adds a certain flexibility.

The flexibility that lies in words and statements we employ is made use of widely in our society. In most western societies, the legal system consists of a certain number of laws. that each describes a different situation. As not for each "real" case a specific law exists, the judge has to combine all applying laws to derive a fair decision. This is only possible due to the flexibility in the definition of the words and statements used in each law.

5.4 Fuzzy Logic as Human Logic

In reality, we cannot define a rule for each possible case. Exact rules (or laws) that cover the respective case perfectly can only be defined for a few distinct cases. These rules are discrete points in the continuum of possible cases and humans approximate between them. Hence, humans combine the rules that describe similar situations. This

approximation is possible due to the flexibility in the definition of the words that constitute the rules.

To implement human logic in engineering solutions, a mathematical model is required. Fuzzy logic has developed such a mathematical model. It allows representing human decision and evaluation processes in algorithmic form. There are limits to what fuzzy logic can do. The full scope of human thinking, fantasy and creativity can not be mimicked with fuzzy logic. However, fuzzy logic can derive a solution for a given case out of rules that have been defined for similar cases. So, if we can describe the desired performance of technical system for certain distinct cases by rules, fuzzy logic will effectively put this knowledge to a solution.

5.5 Membership Functions

The degree to which the value of a technical figure satisfies the linguistic concept of the term of a linguistic variable is called degree of membership. For a continuous variable, this degree is expressed by a function called membership function (MBF). The membership functions map each value of the technical figure to the membership degree to the linguistic terms. The technical quantity is called the base variable. Usually, one draws the membership functions for all terms in the same diagram.

The degree of membership in the figure $\mu(x)$ of the weight x can be represented by a continuous function. Note, that a weight of 184.5 lbs. and a weight of 185.5 lbs. are evaluated differently, but just as a slight bit and not as a threshold.

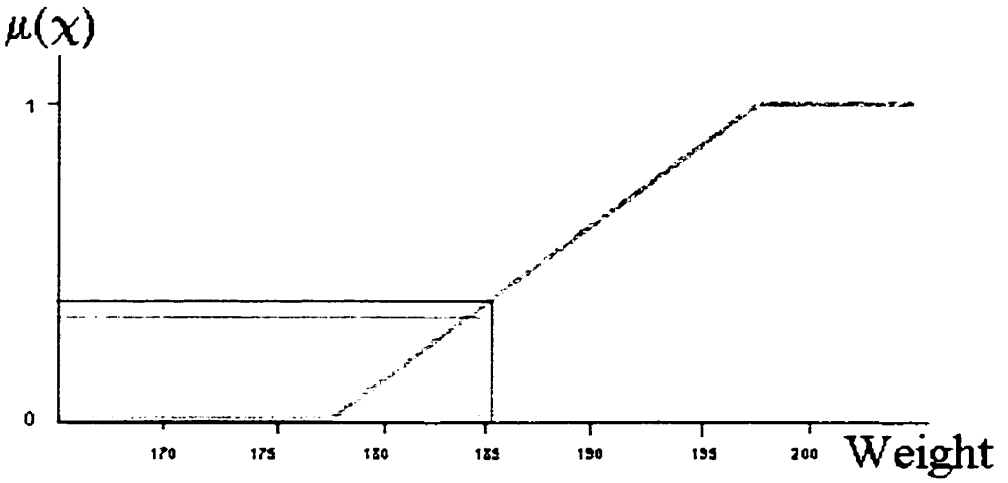

Fig: 13 Degree of Membership

Note that fuzzy sets are a true generalization of conventional sets. The cases μ=0 and μ=1 of the conventional indicator function is just a special case of the fuzzy set. The use of fuzzy sets defined by membership functions in logical expressions is called "fuzzy logic". Here, the degree of membership in a set becomes the degree of truth of a statement. For example, the expression "the passenger is heavy weight" would be true to the degree of 0.65 for a weight of 185 lbs.

5.6 Linguistic Variables

The primary building block of any fuzzy logic system is the "linguistic variable". Here, multiple subjective categories describing the same context are combined. In the case of weight, not only heavy weight but also light weight and medium weight, also exist. These are called "linguistic terms" and represent the possible values of a linguistic variable. The

next figure plots the membership functions of all terms of the linguistic variable fever into the same graph.

A linguistic variable translates real values into linguistic values. This linguistic variable now allows for the translation of a measured body temperature, given in Fahrenheit, into its linguistic description. For example, a body temperature of 190 lbs. would be evaluated as "pretty much heavy weight".

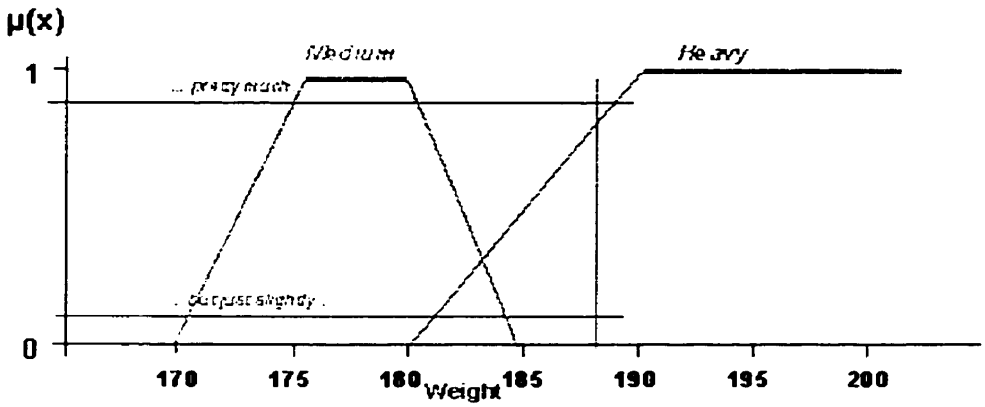

Fig: 14 Defining Linguistic Variable

5.7 Fuzzy Rules

The rules of a fuzzy logic system represent the knowledge of the system. They use linguistic variables as vocabulary, for example to express the control strategy of a fuzzy logic controller. Explaining Fuzzy Rules means to show how to calculate with linguistic concepts. Thus fuzzy rules are explained in detail in the next pages.

5.8 Computation of Fuzzy Logic Systems

5.8.1 Fuzzification

Fuzzification means using the Membership Functions of Linguistic Variables to compute each term's degree of validity at a specific operation point of the process.

Example:

Let "Distance" = 22 inches. The result of fuzzification would be:

far	degree of validity = 0.1
medium	degree of validity = 0.9
low	degree of validity = 0.0
zero	degree of validity = 0.0

Linguistically, a distance of 22 inches could be expressed as almost medium, just slightly far. Fuzzification is the first step in the computation of a fuzzy system and must be performed for each input variable.

The result of fuzzification is used as input for the Fuzzy Rules

5.8.2 Fuzzy Rules

Most fuzzy-based systems use production rules to represent the relation among the linguistic variables and to derive actions from the inputs. Production rules consist of a condition (IF-part) and a conclusion (THEN-part). The IF-part can consist of more than one precondition linked together by linguistic conjunctions like AND and OR.

5.8.3 Fuzzy Rule Inference

The computation of fuzzy rules is called fuzzy rule inference. The inference is a calculus consisting of two main steps: aggregation and conclusion.

Example Rule 1:

IF "Distance" = medium AND "Weight" = Light THEN "Power" = Lvl2

Example Rule 2:

IF "Distance" = far AND "Weight" = zero THEN "Power" = Zero

Example Rule 3:

IF "Distance" = medium AND "Weight" = zero THEN "Power" zero

These fuzzy production rules consist of two preconditions linked together by an AND. The first step of fuzzy inference -- aggregation -- determines the degree to which the complete IF-part of the rule is fulfilled. Special fuzzy operators are used to aggregate the degrees of validity of the various preconditions.

The heart of a fuzzy controller is the list of fuzzy rules. Fuzzy logic inference is used to find a fuzzy output, given a fuzzy input and a list of fuzzy rules. In a fuzzy controller the inputs are normally crisp, nonfuzzy values that must first be fuzzified. The output also needs to be a crisp value used to control some device. Therefore, the fuzzy output resulting from processing the fuzzy rules must be defuzzified. The way fuzzy rules are

processed is illustrated in next 2 figures where fuzzy sets are represented by their membership functions ***m***.

Fuzzy inference involves a set of fuzzy rules of the form

If x_1 is A_1 and x_2 is B_1, then y is L_1. Rule 1

If x_1 is A_2 and x_2 is B_2, then y is L_2. Rule 2

Given the fact that

x_1 is A' and x_2 is B' Fact

the problem is to find the conclusion

y is L' Conclusion

In this representation of the problem, A_1, A_2, B_1, B_2, A', and B' are input fuzzy sets and L_1, L_2, and L' are output fuzzy sets. Fuzzy reasoning would form the union of the intersection of A' and A_1.

This is interpreted as being the maximum (union) of the minimum (intersection) of the membership functions A' and A_1. In figures below A' is taken to be the singleton fuzzy set $x_1 = a$. In Rule 1, the maximum of the intersection (minimum) of this singleton with A_1 is the value w_1. Similarly, the maximum of the intersection (minimum) of the singleton $x_2 = b$ with B_1 is the value w_2. The fact $x_1 = a$ *and* $x_2 = b$ applied to the antecedent x_1 *is* A_1 *and* x_2 *is* B_1 is interpreted as the intersection (minimum) of w_1 and w_2, that is, w_2 for rule 1. The conclusion of rule 1, y *is* L_1, is found by taking the intersection (T-norm) of w_2 with

L_1. This is normally the minimum operation which would truncate L_1 to the height w_2. However, for fuzzy control it is sometimes advantageous to use a product T-norm for this intersection which would have the effect of multiplying L_1 by w_2 as shown in Rule 1. Thus, Rule 1 will contribute the fuzzy set $w_2 * L_1$ to the conclusion fuzzy set L'. Similarly, Rule 2 in figure will contribute the fuzzy set $w_1 * L_2$ to the conclusion fuzzy set L' because w_1 is the minimum of w_1 and w_2 for Rule 2. Note that if L_1 and L_2 are singletons (as is normally the case), then there will be no difference in using the minimum T-norm or the product T-norm.[16]

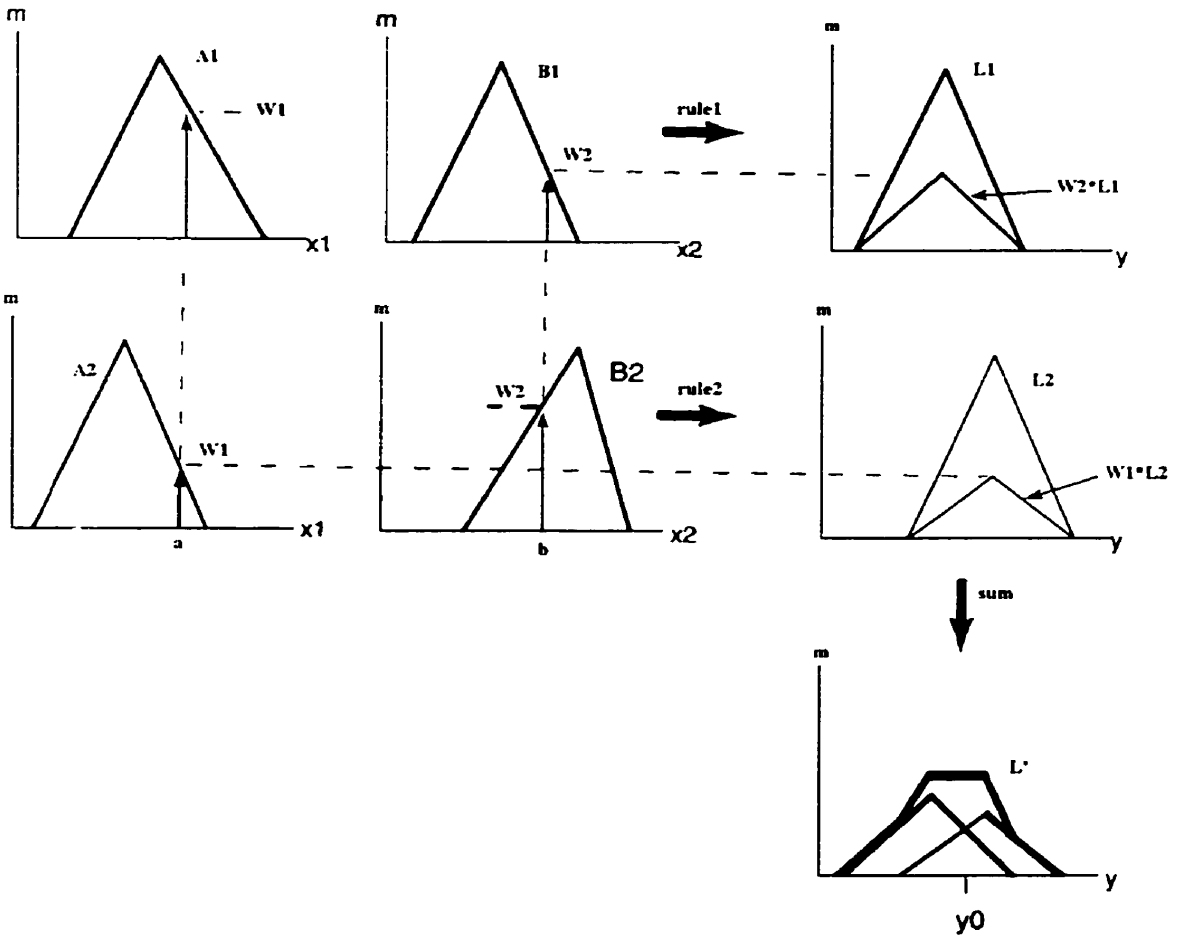

Fig: 15 Fuzzy Interface

The conclusion fuzzy set L' is found by forming the T-conorm of w_2*L_1 and w_1*L_2. This is normally the maximum operation which is the one used by the 68HC12 *REV* instruction as we will see below. However, sometimes better results are obtained by taking the sum of w_2*L_1 and w_1*L_2, as shown in the figure above. The difference between these two approaches is shown in figure below.

If L_1 and L_2 are singletons (the normal case), then taking the maximum or sum of the two rules shown in figure above will be the same as shown in figure below. In general, they won't be the same if more than one rule contribute to the same output fuzzy set L_i. In this case the maximum rule will keep only the maximum value while the sum rule will add the contributions of each. The 68HC12 *REV* instruction uses the maximum rule.

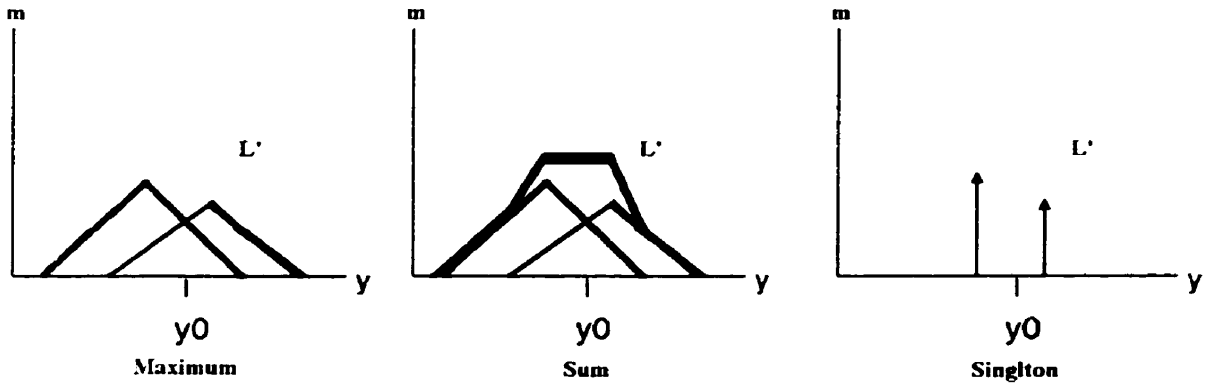

Fig: 16 Comparing The MAX Rule and SUM Rule

The conclusion output L' is a fuzzy set shown by bold-line membership function in both figures. To obtain a crisp output, defuzzification process is required. The most common method is to compute the centroid of the area of L'.

5.8.4 Fuzzy Operators

Operators AND for the minimum and OR for the maximum are often appropriate in small control applications, but sometime other kinds of operators such as MIN and MAX operators, GAMMA or MIN-AVG are needed to signify the relationship of the different parts of the condition.

5.9 MAX-MIN Inference

The second calculation step of each production rule -- composition -- uses the validity of the condition to determine the validity of the conclusion. In standard MAX-MIN or MAX-PROD (sometimes called MAX-DOT) inference methods, the conclusion of a rule is considered equally as true as the conclusion.

5.9.1 FAM Inference

Using standard MAX-MIN/MAX-PROD methods, rule base optimization often consists of arbitrary rule addition/deletion. This method can result in a clumsy trial-and-error approach as the individual importance of a rule can be expressed only as a 0 or 1.

If more than one rule produces the same conclusion (e.g. "Power" = small), the maximum degree of validity of the conclusions is selected for all further processing.

5.9.2 Rule Design

At start with fuzzy technologies, rules with a degree of support of only 0 and 1 (equivalent to MAX-MIN-/MAX-PROD inference) and if individual weighting of rules during optimization is needed then degrees of support between 0 and 1 is used.

Inference methods like MAX-MIN and MAX-PROD are almost same in the beginning of computation but differ from each other when the fuzzy rules result is mapped to the output membership functions.

The first step of computation in both methods is the maximum operation (i.e., the most valid rule is chosen for the final result if different rules result in the same output term).

The second step combines the output values with the output variable's membership functions and clipping (MAX-MIN) or scaling (MAX-PROD) is used for inference.

Scaling: - Membership functions of all terms respective to their degree of validity are multiplied.

Clipping:-, Minimum of membership degree and the fuzzy result of the inference is computed.

These inference methods can be used more-or-less interchangeably, depending on defuzzification method used and they do not differ if we use the Center-of-Maximum defuzzification method.

At the end of the total inference process, all system output variables are associated with a fuzzy value.

5.9.3 Rule Definition

Actual system behavior is defined in the individual rules of the fuzzy system. To prototype an appropriate set of rules, we begin by creating rules which represent unambiguous controller strategies at specific operation points. Once these rules have been established, a step-by-step construction of the rule set can proceed.

Using the matrix rule editor, the rules of a fuzzy system are established with the following steps.

Select the first output variable on the upper axis (horizontal) of the rule matrix. Select the input variable with the most influence on the system on the left axis (vertical) of the rule matrix.

For each combination of input variables not selected on the left axis (select them in the list boxes), find the term which best suits the output variable. Define only those rules with a degree of truth of 0 or 1.

Repeat second step for all output variables on the horizontal axis.

For some combinations of input variables, there is no one exact term, which expresses the desired output value. In this case, do not change the membership function definitions. Instead, use FAM to express the ambiguities.

5.9.4 FAM Rules

If a unique conclusion for a given combination of input variables cannot be found, FAM can be used to express ambiguities.

One approach is to define a new term for mostly zero but somewhat nearly zero. This approach, however, could result in an excessive amount of terms and membership functions. A further drawback is that system structure becomes unnecessarily complex and difficult to survey.

5.10 Uniqueness of a Solution

The goal of fuzzy development is to determine a good solution, which fulfils the technical requirements for the process behavior. But because fuzzy systems are heuristic solutions to real-world technical problems, there are always multiple solutions.

The result of fuzzy rule inference is used as input for Defuzzification.

5.10.1 Defuzzification

The result produced from the evaluation of fuzzy rules is, of course, fuzzy. Naturally, a machine cannot interpret a linguistic command. Membership functions are used to retranslate the fuzzy output into a crisp value. This re-translation is known as defuzzification and can be performed using several methods.[6]

CoM The Center of Maximum method is used for most fuzzy logic applications.

Fast CoA The Center of Area method is similar to CoM and the

MoM Mean of Maximum defuzzification method is used for pattern recognition applications.

MoM BSUM and CoA BSUM are variants of MoM and CoA, which have been optimized for efficient VLSI implementation.

Hyper CoM is used for fuzzy applications, for which not only positive experience in the form of recommendations is of importance, but also negative experience in the form of warnings and prohibitions.

The result of the fuzzy logic inference is the value of a linguistic variable. The conversion of a linguistic result to a real value representing the outcome is called defuzzification.

5.10.1.1 Requirements for Defuzzification Methods

The objective of a defuzzification method is to derive a non-fuzzy (crisp) value, that best contains the fuzzy value of the linguistic output variable. Similar to the different membership function types, different methods for defuzzification exist. To select the proper defuzzification method, we need to understand the linguistic meaning that underlies the defuzzification process.

One defuzzification method to find the best compromise is the Center-of-Maximum. CoM first determines the most typical value for each term and then computes the best compromise of the fuzzy logic inference result. To obtain the best compromizing value for the result of the fuzzy logic inference as a real number, the inference results are considered "weights" at the positions of the most typical values of the terms. The best compromise is where the defuzzified (crisp) value balances the weights.

In some cases, this defuzzification approach does not work. In such cases, the result of the fuzzy logic inference is that no evidence exists. If we would use the Center-of-

Maximum method for defuzzification, a compromise of between two good solutions can lead to a bad result.

Here, the best compromize is clearly not the method of choice. In example we rather want the "most plausible result". One defuzzification method that delivers the "most plausible result" is the "Mean-of-Maximum" method MoM. Rather than balancing out the different inference results, MoM selects the typical value of the term that is most valid.

5.10.1.2 Center-of-Area Defuzzification Method

The first closed-loop control application of fuzzy logic uses a different defuzzification, the so-called Center-of-Area (CoA) method, sometimes called Center-of-Gravity. This method first cuts the membership function at the degree of validity of the respective term. The areas under the resulting functions of all terms are then superimposed. Balancing the resulting area gives the compromising value.

There are some implausibilities with the Center-of-Area method. Another disadvantage of the Center-of-Area defuzzification method is its high computational effort. The center of area is computed by numerical integration that can take up to 1000 times longer than the computation of the center of maximum, depending on the resolution and type of processor. For these reasons, most software development tools and fuzzy logic processors use an approximation of CoA, the so-called fast-CoA. Fast-CoA computes the individual areas under the membership functions during compilation to avoid numerical integration during run time. This approach neglects the overlapping of the areas. Hence it is only an approximation of the "real" CoA. [37]

There are also variants of the Mean-of-Maximum defuzzification method. They differ from MoM by the computation of the most typical value of a membership function.

5.10.2 Continuity of Defuzzification

If an arbitrary small change of an input variable can never cause an abrupt change in any output variable. Then defuzzification method is continuous.

CoM and CoA/CoG methods are continuous while MoM/LoM/RoM are discontinuous. This is due to the fact that the "best compromise" can never jump to a different value for a small change of the inputs. On the other hand, there is always a point where the "most plausible solution" jumps to a different value. There will be a point, where an arbitrary small change in the inputs will cause the decision to turn to the other side.

5.10.3 Defuzzification Method Selection

The continuity property is important for most closed-loop control applications. If the output of a fuzzy logic system directly controls a variable of the process, jumps in the output variable of a fuzzy logic controller can cause instabilities and oscillations. Hence, most closed-loop control applications use CoM defuzzification. Only when the output of the fuzzy logic system proceeds to an integrator first, MoM is a possible alternative. In this case, the integrator keeps the control variable continuous.

Pattern recognition applications mostly use MoM defuzzification. If we want to identify objects by classification of a sensor signal, we are interested in the most plausible result. Some applications even do not use any defuzzification at all. The vector of membership

degrees for the output linguistic variable is the result of the classification as it gives the similarity of the signal to the objects.

In decision support systems, the choice of defuzzification method depends on the context of the decision. Use CoM for quantitative decisions, such as budget allocation or project prioritization. Use MoM for qualitative decisions, such as credit card fraud detection or credit worthiness evaluation.

5.10.4 Information Reduction by Defuzzification

Mathematically, defuzzification is the mapping of a vector (value of the linguistic variable) to a real number (crisp value). This mapping is not unique, that is, different values of a linguistic variable can map to the same defuzzified crisp value.

In practical applications, the only difference between defuzzification methods is, whether they deliver the best compromise (CoM, CoA, and CoG) or the most plausible result (MoM, LoM, and RoM).[

Within these groups, no relevant differences exist that cannot be equalized by modifying membership functions or rules. Complex membership function shapes do not deliver better results for output variables. CoM and MoM defuzzification methods only use the maximum of the membership functions anyway.

In closed-loop control, only use CoM defuzzification. Exceptions are, if the output of the fuzzy logic system proceeds to an integrator.

The wide spread use of CoA/CoG defuzzification has historical reasons. Depending on the overlap and different areas of the membership functions, CoA/CoG can deliver implausible results. Use CoM instead.

Some applications use CoA defuzzification with singleton membership functions. This is completely the same as CoM defuzzification with any membership function type.

Fast-CoA that is used in most software tools and a fuzzy logic processor is equal to a weighted CoM defuzzification.

5.11 Testing and Simulation

5.11.1 Off-Line Optimization

The next step in development process is to simulate and test the prototype designed. We either use pre-recorded data from the process or a process simulation written in a programming language. All techniques used in the second development step are off-line, that is, we work on the PC with no connection to a process in real-time.[18]

5.11.2 On-Line Optimization

For many closed loop control systems we cannot use simulation techniques because no good mathematical model for the process exist. The use of pre-recorded process data is of limited use, as the reaction of the system in real-time to the fuzzy logic controller output is not feed back into the process. In this case, we can use on-line optimization techniques that support "on-the-fly" modifications on a running system.[18]

CHAPTER 6

IMPLEMENTATION

We can implement the fuzzy logic system on target hardware platform after completion. Depending on the target hardware, different implementation techniques exist.

6.1 FUZZY LOGIC AND 68HC12 SUPPORT

A fuzzy inference kernel for the 68HC12 requires one-fifth as much code space, and executes fifteen times faster than a comparable kernel implemented on a typical midrange microcontroller

The 68HC12 includes four instructions that perform specific fuzzy logic tasks. In addition, several other instructions are especially useful in fuzzy logic programs. The overall C-friendliness of the instruction set also aids development of efficient fuzzy logic programs.

The four fuzzy logic instructions are MEM, which evaluates trapezoidal membership functions; REV and REVW, which perform unweighted or weighted MIN-MAX rule evaluation; and WAV, which performs weighted average defuzzification on singleton output membership functions.

Other instructions that are useful for custom fuzzy logic programs include MINA, EMIND, MAXM, EMAXM, TBL, ETBL, and EMACS. For higher resolution fuzzy programs, the fast extended precision math instructions in the 68HC12 are also

beneficial. Flexible indexed addressing modes help simplify access to fuzzy logic data structures stored as lists or tabular data structures in memory. A microcontroller based fuzzy logic control system has two parts.[5], [40], [42]

The first part is a fuzzy inference kernel which is executed periodically to determine system out-puts based on current system inputs. The second part of the system is a knowledge base which contains membership functions and rules.

The knowledge base can be developed by an application expert without any microcontroller programming experience. Membership functions are simply expressions of the expert's understanding of the linguistic terms that describe the system to be controlled. Rules are ordinary language statements that describe the actions a human expert would take to solve the application problem.

Rules and membership functions can be reduced to relatively simple data structures (the knowledge base) stored in nonvolatile memory. A fuzzy inference kernel can be written by a programmer who does not know how the application system works. The only thing the programmer needs to do with knowledge base information is store it in the memory locations used by the kernel.

The design process begins by associating fuzzy sets with the input and output variables. These fuzzy sets are described by membership function of the type shown in figure below. These fuzzy set values are labeled. The shape of the membership functions are, in general, trapezoids that may have no top (triangles) or may have no vertical sides.

A functional diagram of a fuzzy controller is shown in the following figure.

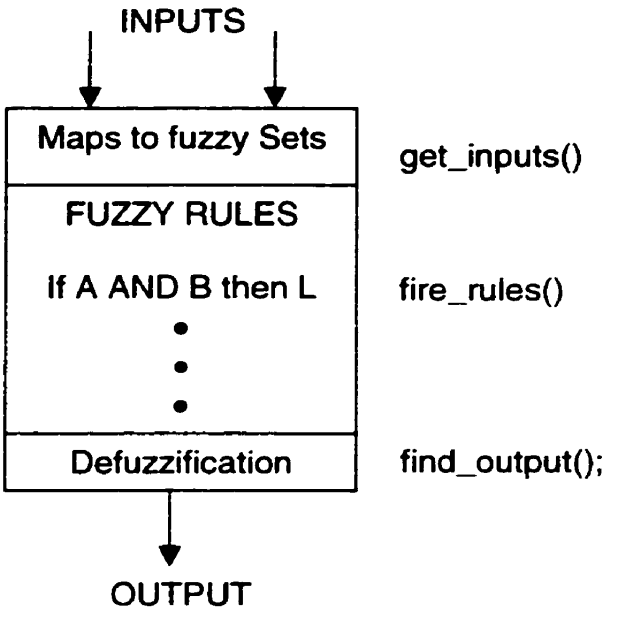

Fig: 17 Functional Diagram of a Fuzzy Controller

The fuzzy controller shown above consists of three parts. The fuzzification of inputs, the processing of rules, and the defuzzification of the output. The inputs to a fuzzy controller are assigned to the fuzzy variables with a degree of membership given by the membership functions. After applying all of the fuzzy rules to a given set of input variables, the output will belong to more than one fuzzy set with different weights. The weighted output fuzzy sets are combined in a manner to be described below and then a centroid defuzzification process is used to obtain a single crisp output value.

Following figure is a block diagram of fuzzy logic system.

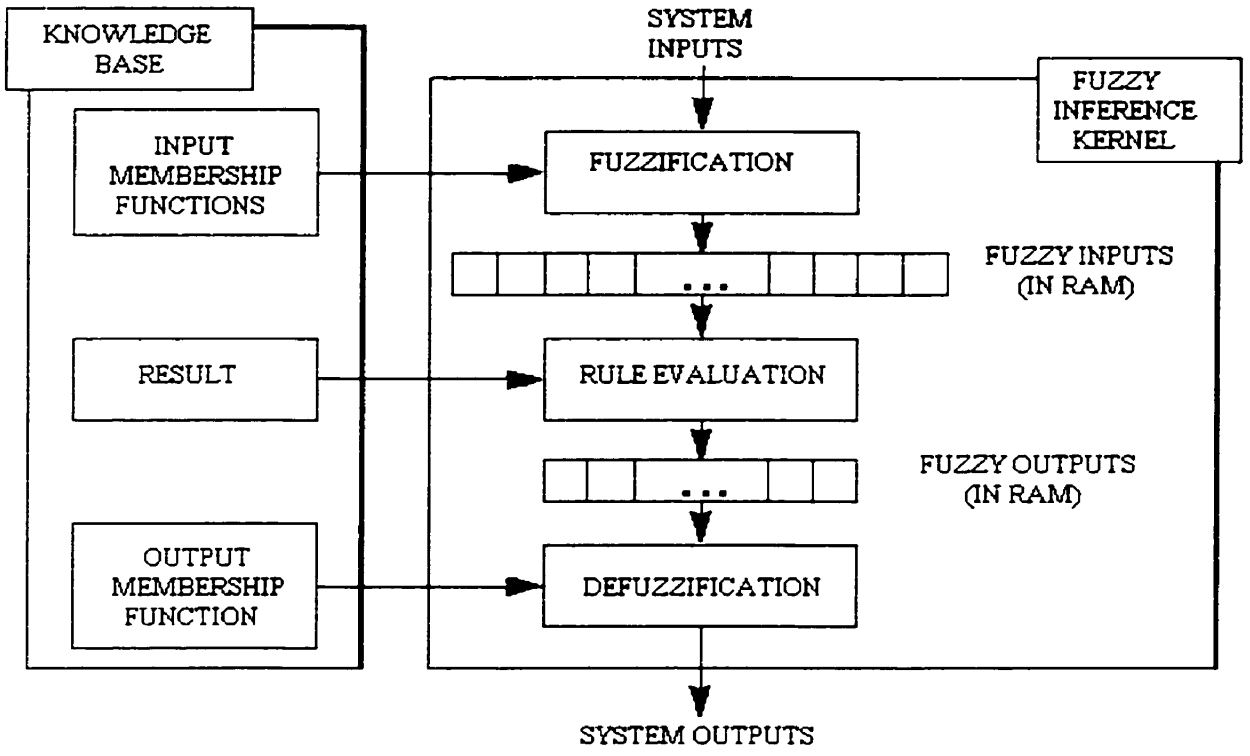

Fig: 18 Block Diagram of Fuzzy Logic System

6.2 Fuzzification of inputs

During the fuzzification step, the current system input values are compared against stored input membership functions to determine the degree to which each label of each system input is true. This is accomplished by finding the y-value for the current input value on a trapezoidal membership function for each label of each system input. The MEM instruction in the 68HC12 performs this calculation for one label of one system input. To perform the complete fuzzification task for a system, several MEM instructions must be executed, usually in a program loop structure. There is a RAM location for each fuzzy input i.e., for each label of each system input.

6.2.1 MEM Instruction

When the fuzzification step begins, the current value of the system input is in an accumulator of the 68HC12, one index register points to the first membership function definition in the knowledge base, and a second index register points to the first fuzzy input in RAM. As each fuzzy input is calculated by executing a MEM instruction the result is stored to the fuzzy input and both pointers are updated automatically to point to the locations associated with the next fuzzy input. The MEM instruction takes care of everything except counting the number of labels per system input and loading the current value of any subsequent system inputs.[16], [38], [39]

One execution pass through the fuzzy inference kernel generates system output signals in response to current input conditions. The kernel is executed as often as needed to maintain control. If the kernel is executed more often than needed, processor bandwidth and power are wasted; delaying too long between passes can cause the system to get too far out of control. Choosing a periodic rate for a fuzzy control system is the same as it would be for a conventional control system.

Each membership function can be defined by the four parameters *u1*, *u2*, *u3*, and *u*4, shown in Figure. The *MEM* instruction requires that the values *u1* and *u4* be 8-bit values between *$00* and *$FE*. The weight values also range from *$00* to *$FF* where *$FF* represents a weight value of 1.0

The *MEM* instruction does not use the parameters *u1*, *u2*, *u3*, and *u4* to define the membership function, Rather it uses *u1* (*point_1*) and *u4* (*point_2*) together with the values of the two slopes, *slope_1* and *slope_2*.

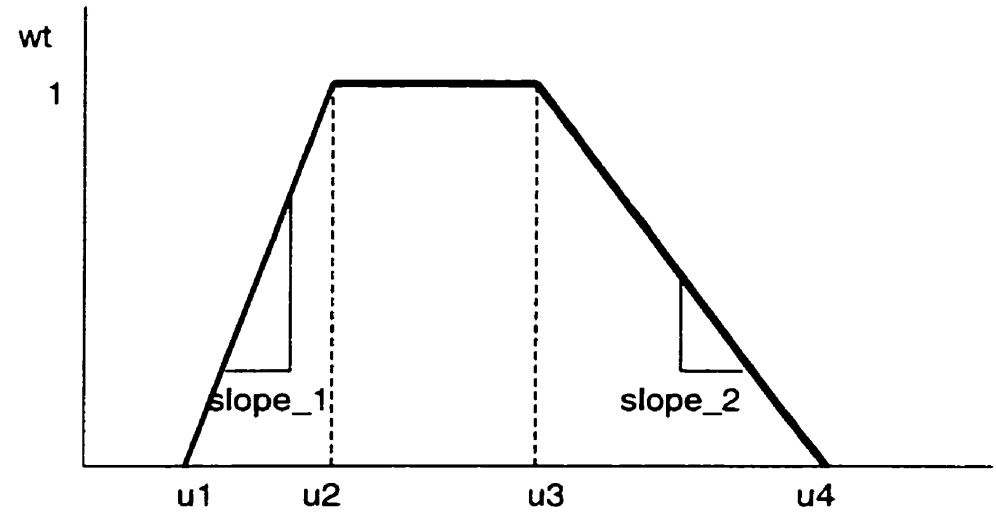

Fig: 19 Defining a Membership Function

The value *of slope_1* is *$FF/(u2 - u1)* and the value of *slope2* is *$FF/(u4 - u3)*. These values can range from *$01* to *$FE*. If *u1* = *u2* or *u3* = *u4* then the slope is really infinite. In this case the values of *slope_1* and/or *slope_2* are taken to be *$00* in as much as this value is not used otherwise. A special case is a singleton, or "crisp," membership function. This can be defined by setting u1 = u4 and *slope_1* = *slope_2* = *$00*.

The *MEM* instruction requires accumulator *A* to contain the input value x_i and index register *X* to point to a data structure containing the two points and slopes that define the membership function. Index register *Y* points to the element of the array corresponding to membership function.[5], [40], [42]

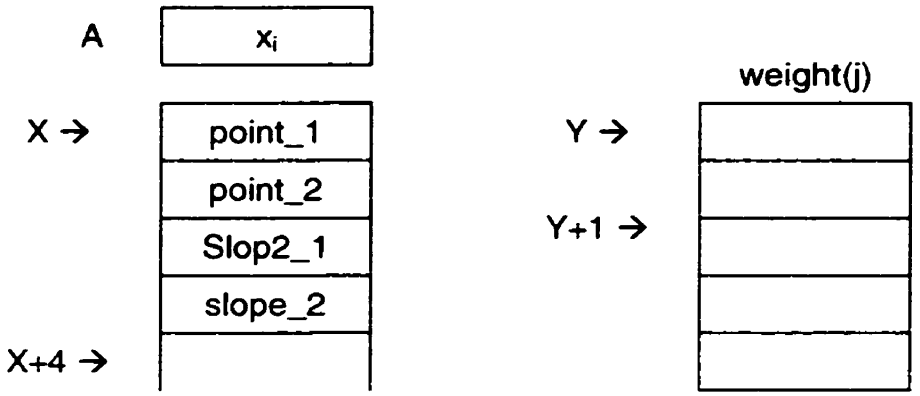

Fig: 20 Data Structure Used by the 68HC12 MEM Instruction

The *MEM* instruction will compute the weight value at the input value x_i based on the membership function whose parameters are pointed to by *X*. The computed weight value *(SOO—SFF)* is stored in the byte pointed to by *Y*. After the *MEM* instruction is executed *X* will have been incremented by 4 and *Y* will have been incremented by 1. If the four parameters of all membership functions for a single input are stored in adjacent bytes of memory, then *X* will he pointing to the parameters of the next membership function. Similarly, *Y* will be pointing to the next element in the array.[16], [[38], [39]

Suppose that an input has the four membership functions then we can store the parameters associated with these four membership functions in the data structure shown below.

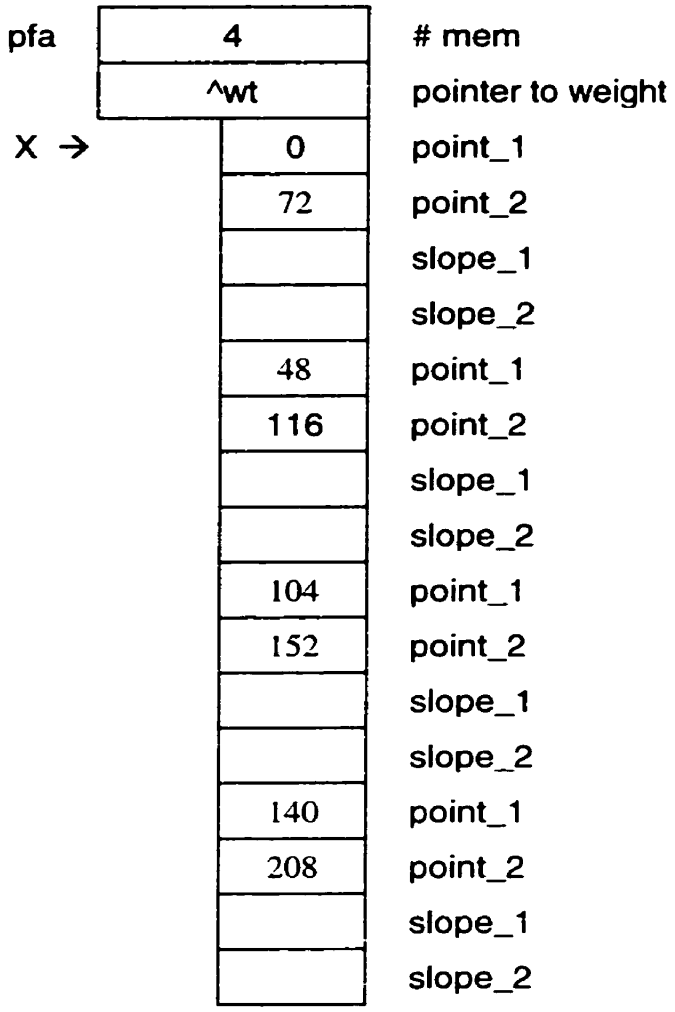

Fig: 21 Data Structure for Storing Membership Function Parameters

The first 16-bit word at address *pfa* contains the number of membership functions. The second 16-bit word contains the address of the array. The next 16 bytes contain the 4-byte parameters for each of the five membership functions. Values for each membership function are indicated by the *point_1* and *point_2*. The *slope_1* and *slope_2* values have been left empty. A subroutine will compute these values and store them in memory.

Note that the *pfa* is first popped from the data stack into *Y* and the value of *X*, is popped from the data stack into *B*. The value of *Y* is transferred to *X* and then loaded with the address of the array. The next three statements will load the number of membership functions into *B*, store x_i in *A*, and leave *X* pointing to the first byte of the parameters associated with the first membership function. This is the setup needed for the *MEM* instruction to execute.

The *MEM* instruction is then executed *B* times (four in this example) by using the looping instruction *DBNE B,FW1*. This instruction will decrement *B* and branch to *FW1* if *B* is not equal to zero. The net result will be that the five weight values associated with the five membership functions for this particular input value, x_i, will be stored in the array.

The end result of the fuzzification step is a table of fuzzy inputs representing current system conditions.

6.3 Rule Evaluation

Rule evaluation is the central element of a fuzzy logic inference program. This step processes a list of rules from the knowledge base using current fuzzy input values from RAM to produce a list of fuzzy outputs in RAM. These fuzzy outputs can be thought of as raw suggestions for what the system output should be in response to the current input conditions. Before the results can be applied, the fuzzy outputs must be further processed, or defuzzified, to produce a single output value that represents the combined effect of all of the fuzzy outputs.

6.3.1 Instructions for Fuzzy Inference

The 68HC12 offers two variations of rule evaluation instructions. The REV instruction provides for unweighted rules (all rules are considered to be equally important). The REVW instruction is similar but allows each rule to have a separate weighting factor which is stored in a separate parallel data structure in the knowledge base. In addition to the weights, the two rule evaluation instructions also differ in the way rules are encoded into the knowledge base.

Complete rules are stored in the knowledge base as a list of pointers or addresses of fuzzy inputs and fuzzy outputs. In order for the rule evaluation logic to work, there needs to be some means of knowing which pointers refer to fuzzy inputs, and which refer to fuzzy outputs. There also needs to be a way to know when the last rule in the system has been reached.

Method of organization used in the 68HC12, is to mark the end of the rule list with a reserved value, and separate antecedents and consequents with another reserved value. This permits any number of rules, and allows each rule to have any number of antecedents and consequents, subject to the limits imposed by availability of system memory.

Each rule is evaluated sequentially, but the rules as a group are treated as if they were all evaluated simultaneously. Two mathematical operations take place during rule evaluation. The fuzzy *and* operator corresponds to the mathematical minimum operation and the fuzzy *or* operation corresponds to the mathematical maximum operation. The

fuzzy *and* is used to connect antecedents within a rule. The fuzzy *or* is implied between successive rules. Before evaluating any rules, all fuzzy outputs are set to zero (meaning not true at all). As each rule is evaluated, the smallest (minimum) antecedent is taken to be the overall truth of the rule. This rule truth value is applied to each consequent of the rule (by storing this value to the corresponding fuzzy output) unless the fuzzy output is already larger (maximum). If two rules affect the same fuzzy output, the rule that is most true governs the value in the fuzzy output because the rules are connected by an implied fuzzy *or*.

6.3.1.1 REV Instruction

Unweighted Rule Evaluation (REV) implements basic *min-max* rule evaluation. X and Y index registers are used as index pointers to the rule list and the fuzzy inputs and outputs. The *accumulator A* is used for intermediate results calculation and must be set to *$FF* initially (the largest 8-bit value). For subsequent rules in the list, *A* is automatically set to *$FF*. when an instruction detects the *$FE* marker character between the last consequent of the previous rule, and the first antecedent of a new rule.[5]. [40], [42]

The *V* condition code bit is used as an instruction status indicator to show whether antecedents or consequents are being processed. Initially, the *V* bit is cleared to zero to indicate antecedents are being processed. The instruction *LDAA #$FF* clears the V bit at the same time it initializes *A* to *$FF*.

The fuzzy outputs (working RAM locations) need to be cleared to *$00* before executing the *REV* instruction.

The *X* index register is set to the address of the first element in the rule list (in the knowledge base). The Y index register is set to the base address for the fuzzy inputs and outputs (in working RAM). Each rule antecedent and consequent are unsigned 8-bit offset from base address to the referenced fuzzy input and fuzzy output respectively.

The 8-bit *accumulator A* is used to hold intermediate calculation results during execution of the *REV* instruction. During this process, *A* starts out at *$FF* and is replaced by any smaller fuzzy input that is referenced by a rule antecedent (*MIN*). During consequent processing, *A* holds the truth value for the rule. This truth value is stored to any fuzzy output that is referenced by a rule consequent, unless that fuzzy output is already larger (*MAX*).

The final requirement to clear all fuzzy outputs to *$00* is part of the *MAX* algorithm. Each time a rule consequent references a fuzzy output, that fuzzy output is compared to the truth value for the current rule. If the current truth value is larger, it is written over the previous value in the fuzzy output. After all rules have been evaluated, the fuzzy output contains the truth value for the most-true rule that referenced that fuzzy output. After *REV* finishes, A will hold the truth value for the last rule in the rule list. The *V* condition code bit should be one because the last element before the *$FF* end marker should have been a rule consequent. If V is zero after executing REV, it indicates the rule list was structured incorrectly.

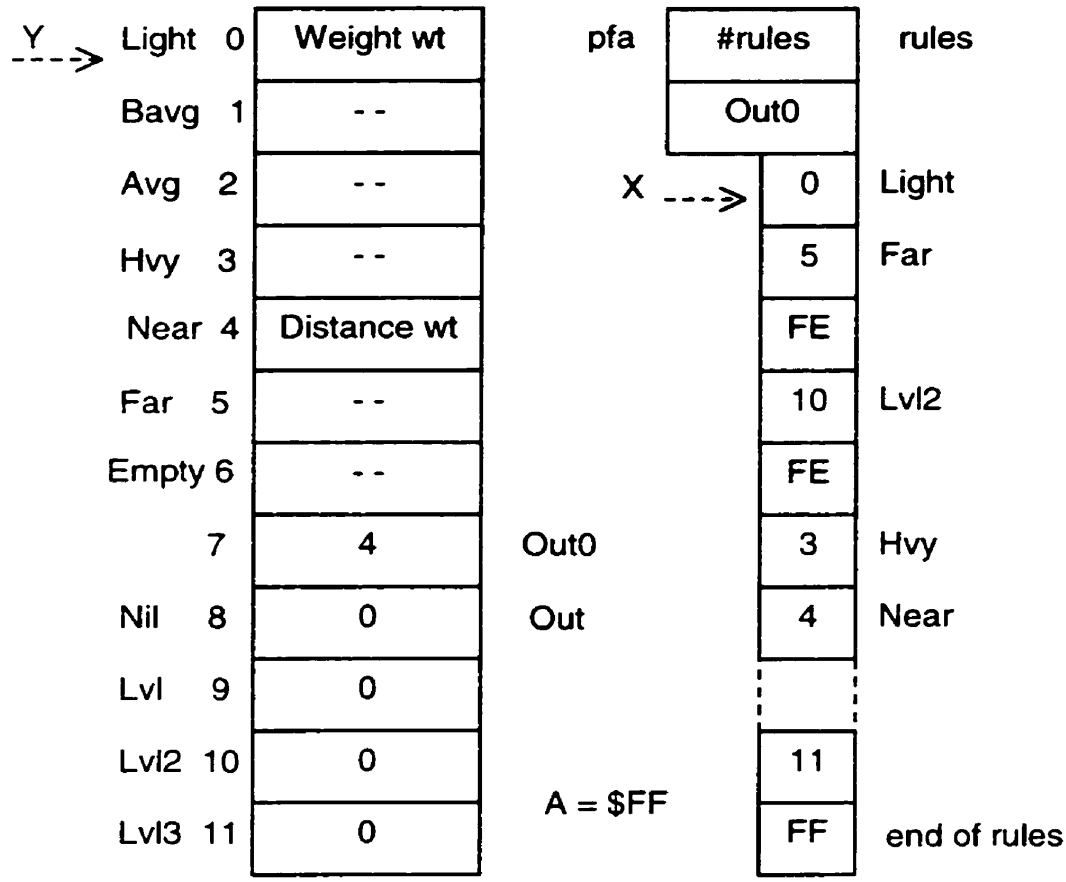

Fig: 22 Setup Required for REV Instruction.[16], [[38], [39]

6.3.1.2 REVW Instruction

The Weighted Rule Evaluation (*REVW*) is a weighted variation of *MIN-MAX* rule evaluation. Before applying the truth value to the consequents for the rule, the value is multiplied by a fraction from zero (rule disabled) to one (rule fully enabled). The resulting modified truth value is then applied to the fuzzy outputs.

The rule structure for *REVW* is made up of *16*-bit elements rather than *8*-bit elements as in *REV* instruction. Each antecedent and consequent is represented by the full *16*-bit address of the corresponding fuzzy input and fuzzy output respectively.

The *16*-bit markers *$FFFE* separates the antecedents from consequents and the end of the last rule is marked by the reserved 16-bit value *$FFFF*.

X and *Y* index registers are used as index pointers to the rule list and the list of rule weights. The *8*-bit accumulator A is used to hold intermediate calculation results during execution of the *REVW* instruction and must be set to *$FF* initially. During antecedent processing, A starts out at *$FF* and is replaced by any smaller fuzzy input that is referenced by a rule antecedent. The *V* condition code bit is used as an instruction status indicator that shows whether antecedents or consequents are being processed. Initially the *V* bit is cleared to zero to indicate antecedents are being processed. The C condition code bit is used to indicate whether rule weights are to be used (1) or not (0). If rule weights are enabled by the C condition code bit equal one, the rule truth value is multiplied by the rule weight just before consequent processing starts. The fuzzy outputs (working RAM locations) is cleared to $00. These values must be initialized before *REVW* instruction is executed to avoid errors in the result.

The *X* index register is set to the address of the first element in the rule list (in the knowledge base). After the *REVW* instruction finishes, *X* will point at the next address past the *$FFFF* separator word that marks the end of the rule list. The *Y* index register is set to the starting address of the list of rule weights.

Each rule weight is an 8-bit value which is driven by multiplying the minimum rule antecedent value (*$00-$FF*) by the weight plus one (*$001-$100*). The weighted result is the truncated upper 8 bits of the 16-bit result,. This method of weighting rules allows an 8-bit weighting factor to represent a value between zero and one inclusive.

During consequent processing, A holds the truth value (possibly weighted) for the rule. This truth value is stored to any fuzzy output that is referenced by a rule consequent, unless that fuzzy output is already larger (*MAX*).Accumulator A is automatically set to *$FF* when the instruction detects the *$FFFE* marker word between the last consequent of the previous rule, and the first antecedent of a new rule.

Once the *REVW* instruction starts, the C bit remains constant and the value in the V bit is automatically maintained as *$FFFE* separator words are detected.

The final requirement to clear all fuzzy outputs to $00 is part of the *MAX* algorithm. Each time a rule consequent references a fuzzy output, that fuzzy output is compared to the truth value (weighted) for the current rule. If the current truth value is larger, it is written over the previous value in the fuzzy output. After all rules have been evaluated, the fuzzy output contains the truth value for the most-true rule that referenced that fuzzy output.

After *REVW* finishes, A will hold the truth value (weighted) for the last rule in the rule list. The V condition code bit should be one because the last element before the *$FFFF* end marker should have been a rule consequent. If *V* is zero after executing *REVW*, it indicates the rule list was structured incorrectly.

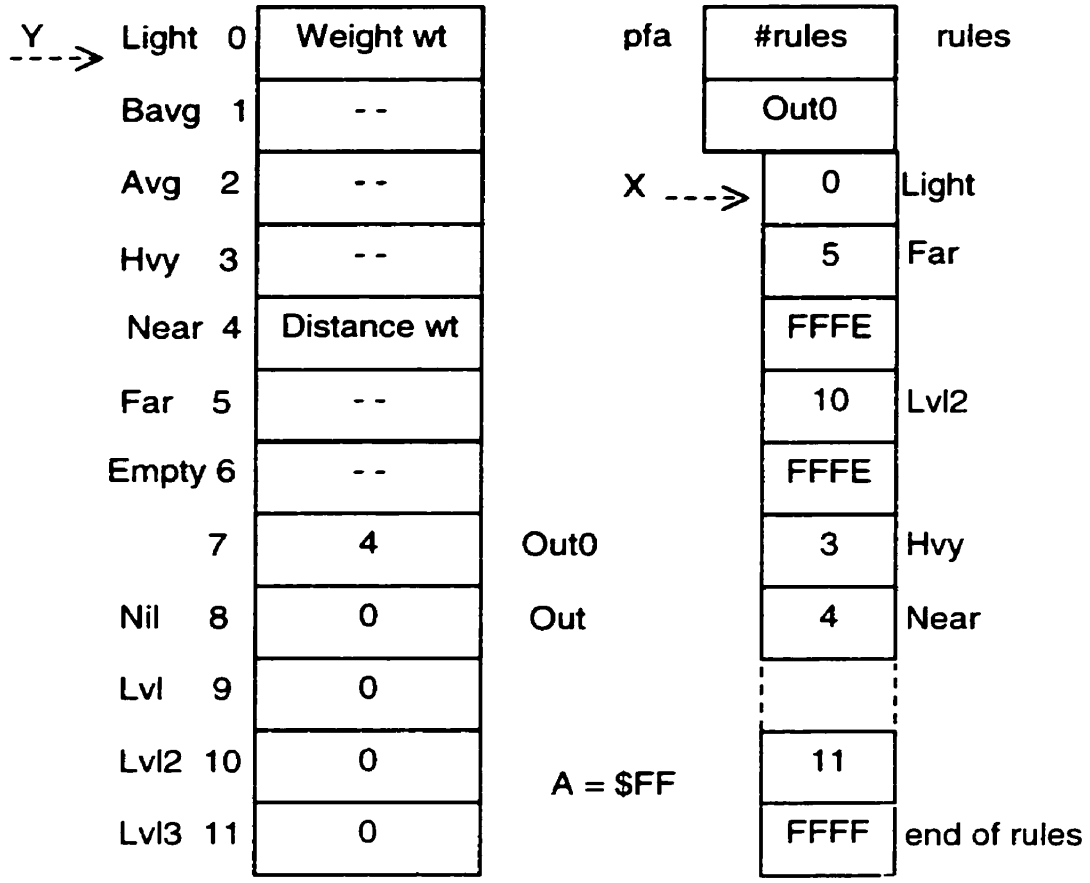

Fig: 23 Setup Required for REVW Instruction.[16], [[38], [39]

6.4 Defuzzification

The end result of the rule evaluation step is a table of suggested or "raw" fuzzy outputs in RAM. These values were obtained by plugging current conditions (fuzzy input values) into the system rules in the knowledge base. The raw results cannot be supplied directly to the system outputs because they may be ambiguous. For instance, one raw output can indicate that the system output should be medium with a degree of truth of 50% while, at the same time, another indicates that the system output should be low with a degree of truth of 25%. The defuzzification step resolves these ambiguities.

The final step in the fuzzy logic program combines the raw fuzzy outputs into a composite system output. Unlike the trapezoidal shapes used for inputs, the 68HC12 typically uses singletons for output membership functions. As with the in-puts, the x-axis represents the range of possible values for a system output. Singleton membership functions consist of the x-axis position for a label of the system output. Fuzzy outputs correspond to the y-axis height of the corresponding output membership function.

The WAV instruction calculates the numerator and denominator sums for weighted average of the fuzzy outputs according to the following formula:

$$y_0 = \frac{\sum_{k=1}^{Q} W^k c^k}{\sum_{k=1}^{Q} W^k}$$

Before executing WAV, an accumulator must be loaded with the number of iterations (n), one index register must be pointed at the list of singleton positions in the knowledge base, and a second index register must be pointed at the list of fuzzy outputs in RAM. If the system has more than one system output, the WAV instruction is executed once for each system output. .[16], [[38], [39]

6.4.1 WAV Instruction

It performs weighted average calculations on values stored in the memory and uses indexed (*X*) addressing mode to reference one source operand list, and indexed (*Y*)

addressing mode to reference a second source operand list. Accumulator B is used as a counter to control the number of elements to be included in the weighted average.

For each pair of data points, a 24-bit Sum Of Products (SOP) and a 16-bit Sum Of Weights (*SOW*) is accumulated in temporary registers. When B reaches zero (no more data pairs), the SOP is placed in *Y:D*. The *SOW* is placed in X.

To arrive at the final weighted average, divide the content of Y:D by X by executing an *EDIV* after the *WAV*. This instruction can be interrupted. If an interrupt occurs during *WAV* execution, the intermediate results (six bytes) are stacked in the order *SOW [15:0]*, *SOP [15:0]*, *$00:SOP [23:16]* before the interrupt is processed.

CHAPTER 7

AIRBAG

7.1 Project Description

Linguistic Input Variables	4
Linguistic Output Variables	1
Intermediate Variables	0
Rule Blocks	1
Rules	72
Membership Functions	16

Project Statistics

7.2 System Structure

The system structure identifies the fuzzy logic inference flow from the input variables to the output variables. The fuzzification in the input interfaces translates analog inputs into fuzzy values. The fuzzy inference takes place in rule blocks which contain the linguistic control rules. The output of these rule blocks are linguistic variables. The defuzzification in the output interfaces translates them into analog variables.

The following figure shows the whole structure of this fuzzy system including input interfaces, rule blocks and output interfaces. The connecting lines symbolize the data flow.

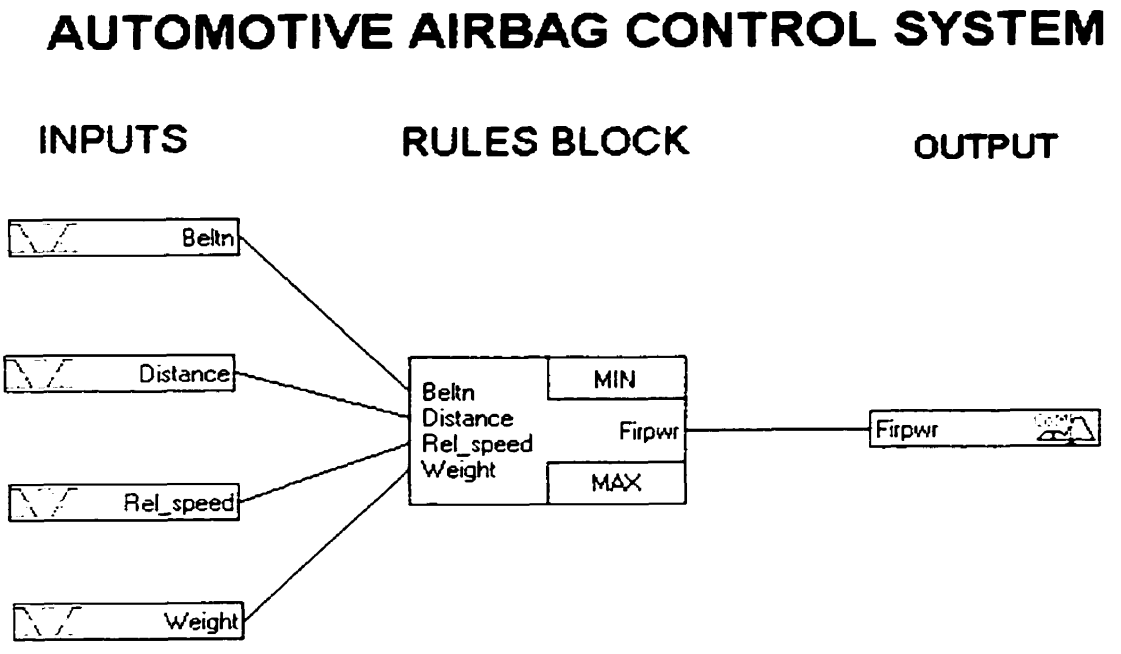

Fig: 24 **Structure of the Fuzzy Logic System**

WEIGHT IN POUNDS IN RELATION TO HEIGHT FOR ADULT

MEN AND WOMEN, 25 YEARS OR OLDER

Men Medium Frame				Women Medium Frame			
Height				Height			
Ft	In.	Range	Midpoint	Ft	In.	Range	Midpoint
				4	8	93---104	98.5
				4	9	95---107	101
				4	10	98---110	104
				4	11	101---113	107
				5	0	104---116	110
5	1	113---124	118.5	5		107---119	113
5	2	116---128	122	5	2	110---123	116.5
5	3	119---131	125	5	3	113---127	120
5	4	122---134	128	5	4	117---132	124.5
5	5	125---138	131.5	5	5	121---136	128.5
5	6	129---142	135.5	5	6	125---140	132.5
5	7	133---147	140	5	7	129---144	136.5
5	8	137---151	144	5	8	133---148	140.5
5	9	141---155	148	5	9	137---152	144.5
5	10	145---160	153	5	0	141---156	148.5
5	11	149---165	157				
6	0	153---170	161.5				
6	1	157---175	166				
6	2	162---180	171				
6	3	167---185	176				

Adopted from the Metropolitan Insurance Company Statistical Bulletin

Table: 1

[24]

AGE-SPECIFIC WEIGHT-FOR-HEIGHT TABLES* (GERONTOLOGY RESEARCH CENTER)					
	Weight Range (lbs) for Men and Women				
Height	by Age (Years)				
(ft in)	25	35	45	55	65
4 10	84__111	92__119	99__127	107__135	115__142
4 11	87__115	95__123	103__131	111__139	119__147
5 0	90__119	98__127	106__135	114__143	123__152
5 1	93__123	101__131	110__140	118__148	127__157
5 2	96__127	105__136	113__144	122__153	131__163
5 3	99__131	108__140	117__149	126__158	135__168
5 4	102__135	112__145	121__154	130__163	140__173
5 5	106__140	115__149	125__159	134__168	144__179
5 6	109__144	119__154	129__164	138__174	148__184
5 7	112__148	122__159	133__169	143__179	153__190
5 8	116__153	126__163	137__174	147__184	158__196
5 9	119__157	130__168	141__179	151__190	162__201
5 10	122__162	134__173	145__184	156__195	167__207
5 11	126__167	137__178	149__190	160__201	172__213
6 0	129__171	141__183	153__195	165__207	177__219
6 1	133__176	145__188	157__200	169__213	182__225
6 2	137__181	149__194	162__206	174__219	187__232
6 3	141__186	153__199	166__212	179__225	192__238
6 4	144__191	157__205	171__218	184__231	197__244

Table: 2

[25]

Distance Between Instrument Panel and Passenger Side Back of the Seat		
Model	**Pushed Forward**	**Pushed Backward**
	Centimeters	
Ford		
Grand Marquis	53	76
Taurus	51	74
Mustang	53	76
Cougar	53	76
Windstar	56	74
General Motors		
GMC Truck	56	76
Envoy	51	71
Montana	56	76
Regal	51	76
Chrysler		
Neon	53	76
Cirrus	51	74
Intrepid	56	79
Caravan	56	79

Table: 3

7.3 Linguistic Variables

The following table lists all linguistic variables of the system and their term names.

Variable Name	Term Names
Beltn	Unbkl, Bkld
Distance	Near, Far, Empty
Rel_speed	Cty, Hiwy, Ospd
Weight	Light, Bavg, Avg, Hvy
Inf_Speed	Nil, Lvl1, Lvl2, Lvl3

Linguistic Variables

The properties of all base variables are listed in the following table.

Variable Name	Min	Max	Default	Unit
Beltn	0	1	1	Units
Distance	50	80	65	Cm
Rel_speed	40	150	95	Kms
Weight	50	100	75	Kgs
Inf_Speed	0	200	100	Kmph

Table: 4 Base Variables

The default value of an output variable is used if no rule is firing for this variable. Different methods can be used for the defuzzification, resulting either in to the 'most plausible result' or the 'best compromise'.

The 'best compromise' is produced by the methods:

CoM (Center of Maximum)

CoA (Center of Area)

CoA BSUM, a version especially for efficient VLSI implementations

The 'most plausible result is produced by the methods:

MoM (Mean of Maximum)

MoM BSUM, a version especially for efficient VLSI implementations

The following table lists all variables linked with an interface as well as the respective fuzzification or defuzzification method.

Variable Name	Type	Fuzzification/Defuzzification
Beltn	Input	Compute MBF
Distance	Input	Compute MBF
Rel_speed	Input	Compute MBF
Weight	Input	Compute MBF
Inf_Speed	Output	CoM

Table: 5 Interfaces

7.3.1 Input Variable "Beltn"

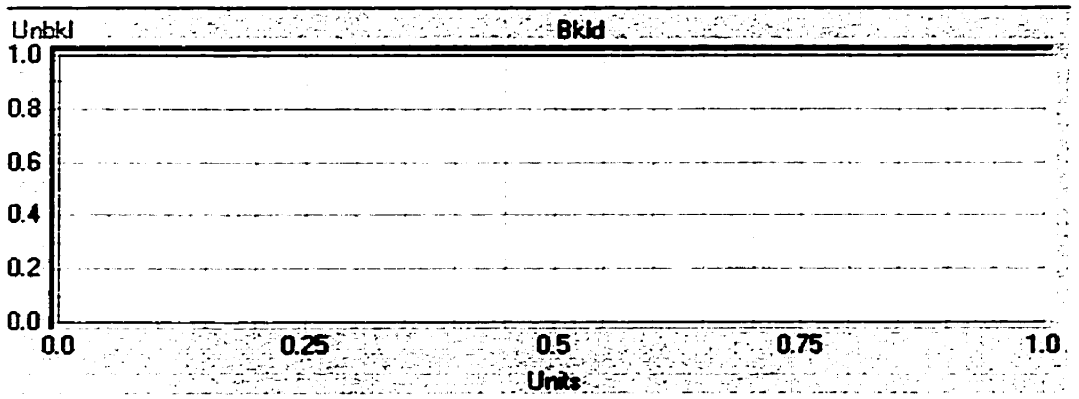

Fig: 25 MBF of "Beltn"

Term Name	Shape/Par.	Definition Points (x, y)		
Unbkl	linear	(0, 1)	(0, 0)	(1, 0)
Bkld	linear	(0, 1)	(0.6, 1)	(1, 1)

Tale: 6 Definition Points of MBF "Beltn"

7.3.2 Input Variable "Distance"

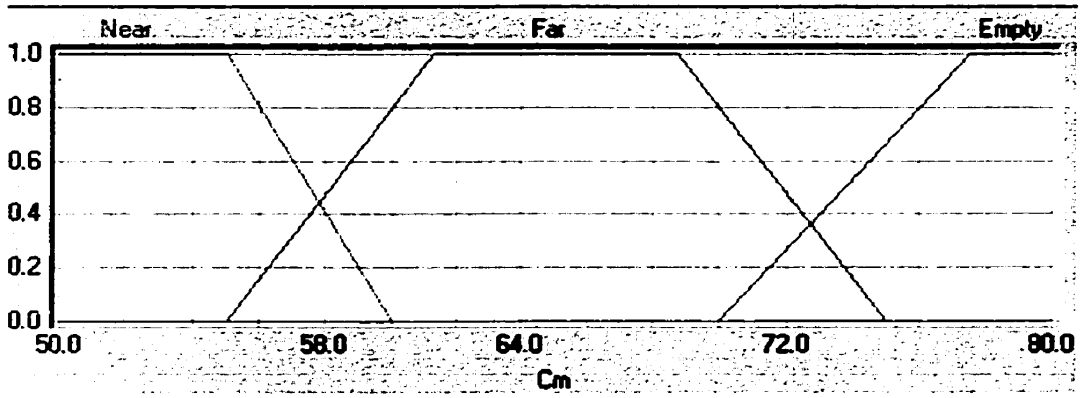

Fig: 26 MBF of "Distance"

Term Name	Shape/Par.	Definition Points (x, y)		
Near	linear	(50, 1)	(55, 1)	(60, 0)
		(80, 0)		
Far	linear	(50, 0)	(55, 0)	(61.25, 1)
		(68.75, 1)	(75, 0)	(80, 0)
Empty	linear	(50, 0)	(70, 0)	(77.5, 1)
		(80, 1)		

Table: 7 Definition Points of MBF "Distance"

7.3.3 Input Variable "Rel_speed"

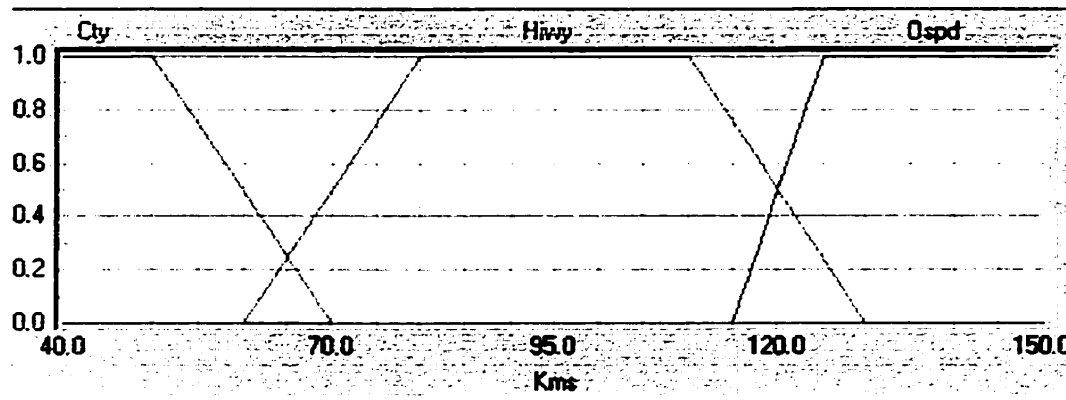

Fig: 27 MBF of "Rel_speed"

Term Name	Shape/Par.	Definition Points (x, y)		
Cty	linear	(40, 1)	(50, 1)	(70, 0)
		(150, 0)		
Hiwy	linear	(40, 0)	(60, 0)	(80, 1)
		(110, 1)	(130, 0)	(150, 0)
Ospd	linear	(40, 0)	(115, 0)	(125, 1)
		(150, 1)		

Table: 8 Definition Points of MBF "Rel_speed"

7.3.4 Input Variable "Weight"

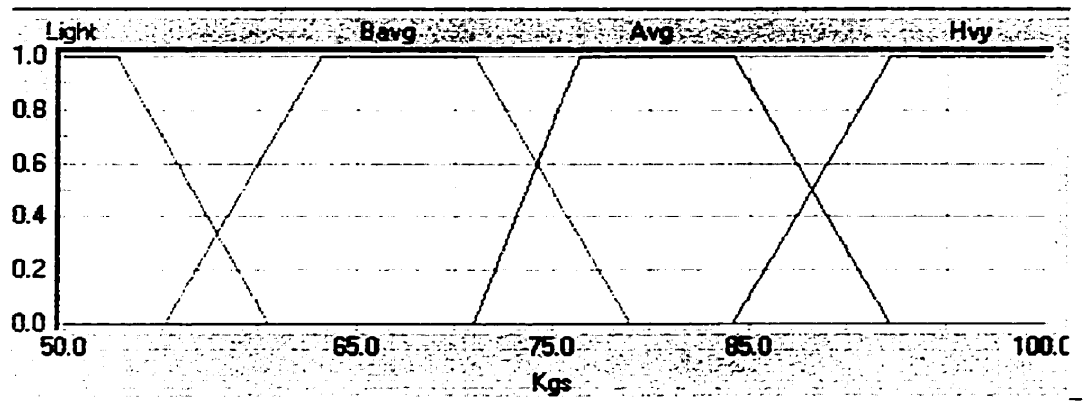

Fig: 28 MBF of "Weight"

Term Name	Shape/Par.	Definition Points (x, y)		
Light	linear	(50, 1)	(52.6, 1)	(60.6, 0)
		(100, 0)		
Bavg	linear	(50, 0)	(55.2, 0)	(63.2, 1)
		(71, 1)	(79, 0)	(100, 0)
Avg	linear	(50, 0)	(71, 0)	(76.4, 1)
		(84.2, 1)	(92.2, 0)	(100, 0)
Hvy	linear	(50, 0)	(84.2, 0)	(92.2, 1)
		(100, 1)		

Table: 9 Definition Points of MBF "Weight"

7.3.5 Output Variable "Inf_Speed"

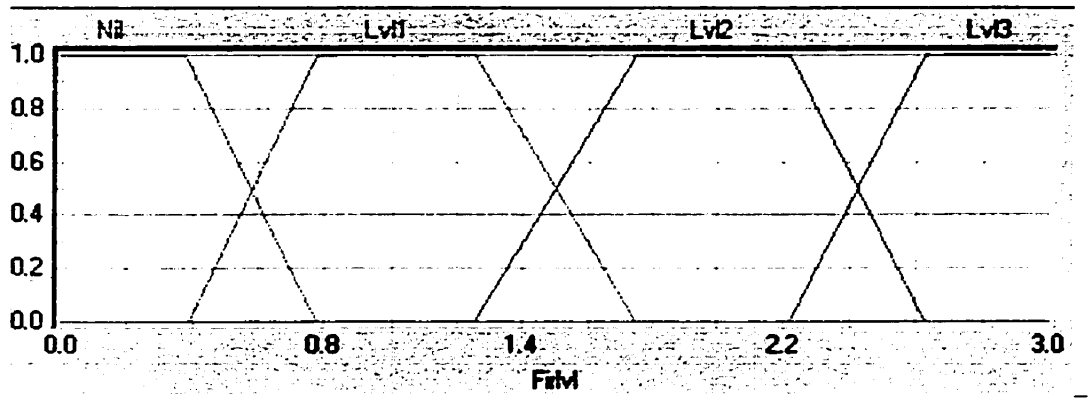

Fig: 29 MBF of "Inf_Speed"

Term Name	Shape/Par.	Definition Points (x, y)		
Nil	linear	(0, 1)	(20, 1)	(60, 0)
		(200, 0)		
Lvl1	linear	(0, 0)	(20, 0)	(50, 1)
		(80, 1)	(120, 0)	(200, 0)
Lvl2	linear	(0, 0)	(80, 0)	(120, 1)
		(150, 1)	(180, 0)	(200, 0)
Lvl3	linear	(0, 0)	(150, 0)	(180, 1)
		(200, 1)		

Table: 10 Definition Points of MBF "Inf_Speed"

7.4 Rule Blocks

7.4.1 Parameter

Aggregation:	MINMAX
Parameter:	0.00
Result Aggregation:	MAX
Number of Inputs:	4
Number of Outputs:	1
Number of Rules:	72

7.7.2 Rules

IF				THEN	
Beltn	**Distance**	**Rel_speed**	**Weight**	**DoS**	**Inf_Speed**
Unbkl	Near	Cty	Light	1.00	Lvl1
Unbkl	Near	Cty	Bavg	1.00	Lvl1
Unbkl	Near	Cty	Avg	1.00	Lvl2
Unbkl	Near	Cty	Hvy	1.00	Lvl2
Unbkl	Near	Hiwy	Light	1.00	Lvl2
Unbkl	Near	Hiwy	Bavg	1.00	Lvl2
Unbkl	Near	Hiwy	Avg	1.00	Lvl2
Unbkl	Near	Hiwy	Hvy	1.00	Lvl2
Unbkl	Near	Ospd	Light	1.00	Lvl2
Unbkl	Near	Ospd	Bavg	1.00	Lvl3

Unbkl	Near	Ospd	Avg	1.00	Lvl3
Unbkl	Near	Ospd	Hvy	1.00	Lvl3
Unbkl	Far	Cty	Light	1.00	Lvl3
Unbkl	Far	Cty	Bavg	1.00	Lvl3
Unbkl	Far	Cty	Avg	1.00	Lvl3
Unbkl	Far	Cty	Hvy	1.00	Lvl3
Unbkl	Far	Hiwy	Light	1.00	Lvl3
Unbkl	Far	Hiwy	Bavg	1.00	Lvl3
Unbkl	Far	Hiwy	Avg	1.00	Lvl3
Unbkl	Far	Hiwy	Hvy	1.00	Lvl3
Unbkl	Far	Ospd	Light	1.00	Lvl3
Unbkl	Far	Ospd	Bavg	1.00	Lvl3
Unbkl	Far	Ospd	Avg	1.00	Lvl3
Unbkl	Far	Ospd	Hvy	1.00	Lvl3
Unbkl	Empty	Cty	Light	1.00	Nil
Unbkl	Empty	Cty	Bavg	1.00	Nil
Unbkl	Empty	Cty	Avg	1.00	Nil
Unbkl	Empty	Cty	Hvy	1.00	Nil
Unbkl	Empty	Hiwy	Light	1.00	Nil
Unbkl	Empty	Hiwy	Bavg	1.00	Nil
Unbkl	Empty	Hiwy	Avg	1.00	Nil
Unbkl	Empty	Hiwy	Hvy	1.00	Nil
Unbkl	Empty	Ospd	Light	1.00	Nil
Unbkl	Empty	Ospd	Bavg	1.00	Nil
Unbkl	Empty	Ospd	Avg	1.00	Nil
Unbkl	Empty	Ospd	Hvy	1.00	Nil

Bkld	Near	Cty	Light	1.00	Nil
Bkld	Near	Cty	Bavg	1.00	Nil
Bkld	Near	Cty	Avg	1.00	Lvl1
Bkld	Near	Cty	Hvy	1.00	Lvl1
Bkld	Near	Hiwy	Light	1.00	Lvl1
Bkld	Near	Hiwy	Bavg	1.00	Lvl1
Bkld	Near	Hiwy	Avg	1.00	Lvl2
Bkld	Near	Hiwy	Hvy	1.00	Lvl2
Bkld	Near	Ospd	Light	1.00	Lvl2
Bkld	Near	Ospd	Bavg	1.00	Lvl2
Bkld	Near	Ospd	Avg	1.00	Lvl2
Bkld	Near	Ospd	Hvy	1.00	Lvl2
Bkld	Far	Cty	Light	1.00	Lvl2
Bkld	Far	Cty	Bavg	1.00	Lvl2
Bkld	Far	Cty	Avg	1.00	Lvl3
Bkld	Far	Cty	Hvy	1.00	Lvl3
Bkld	Far	Hiwy	Light	1.00	Lvl3
Bkld	Far	Hiwy	Bavg	1.00	Lvl3
Bkld	Far	Hiwy	Avg	1.00	Lvl3
Bkld	Far	Hiwy	Hvy	1.00	Lvl3
Bkld	Far	Ospd	Light	1.00	Lvl3
Bkld	Far	Ospd	Bavg	1.00	Lvl3
Bkld	Far	Ospd	Avg	1.00	Lvl3
Bkld	Far	Ospd	Hvy	1.00	Lvl3
Bkld	Empty	Cty	Light	1.00	Nil
Bkld	Empty	Cty	Bavg	1.00	Nil

Bkld	Empty	Cty	Avg	1.00	Nil
Bkld	Empty	Cty	Hvy	1.00	Nil
Bkld	Empty	Hiwy	Light	1.00	Nil
Bkld	Empty	Hiwy	Bavg	1.00	Nil
Bkld	Empty	Hiwy	Avg	1.00	Nil
Bkld	Empty	Hiwy	Hvy	1.00	Nil
Bkld	Empty	Ospd	Light	1.00	Nil
Bkld	Empty	Ospd	Bavg	1.00	Nil
Bkld	Empty	Ospd	Avg	1.00	Nil
Bkld	Empty	Ospd	Hvy	1.00	Nil

Table:11 Rules of the Rule Block "RB1"

7.5 List of Abbreviations

Compute MBF	Compute Membership Function (Fuzzification Method)
CoM	Center of Maximum (Defuzzification Methode)
BSUM	Bounded Sum Fuzzy Operator for Result Aggregation
MIN	Fuzzy Operator for AND Aggregation
MAX	Fuzzy Operator for OR Aggregation
GAMMA	Compensatory Operator for Aggregation
PROD	Fuzzy Operator for Composition
LV	Linguistic Variable
MBF	Membership Function
RB	Rule Block

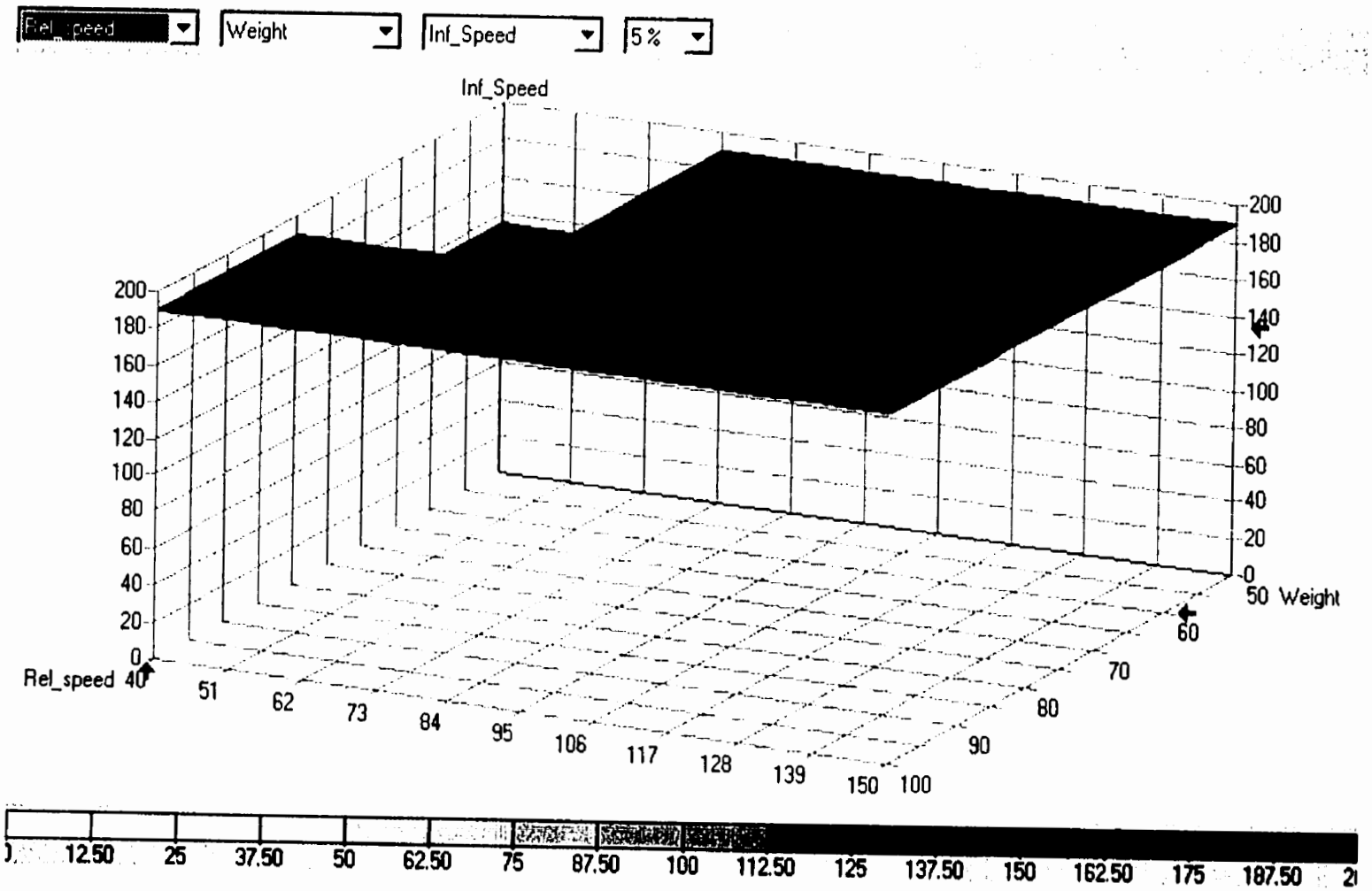

Fig: 30 "Inf_Speed" w.r.t. "Rel_Speed" and "Weight"

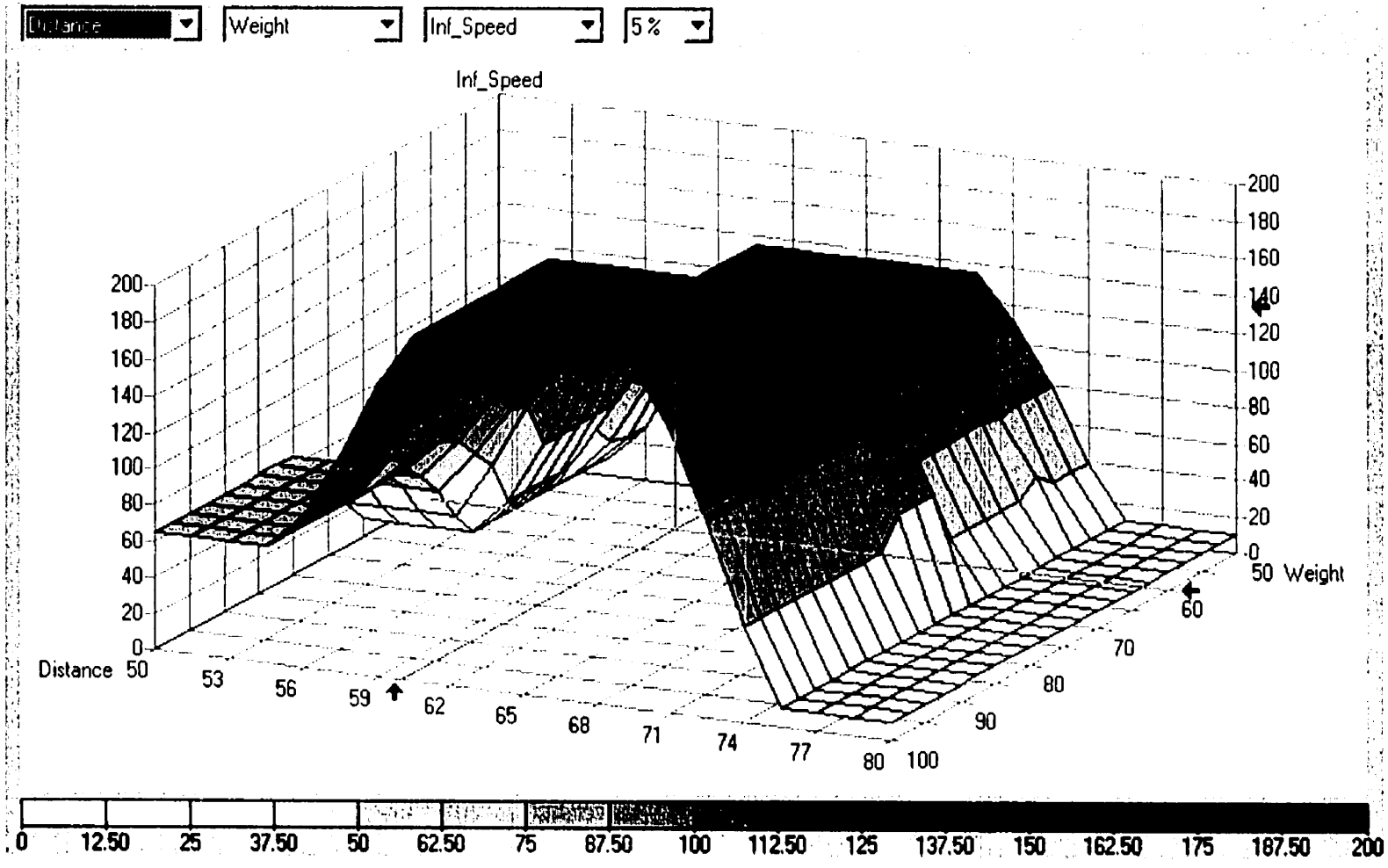

Fig: 31 "Inf_Speed" w.r.t. "Distance" and " Weight"

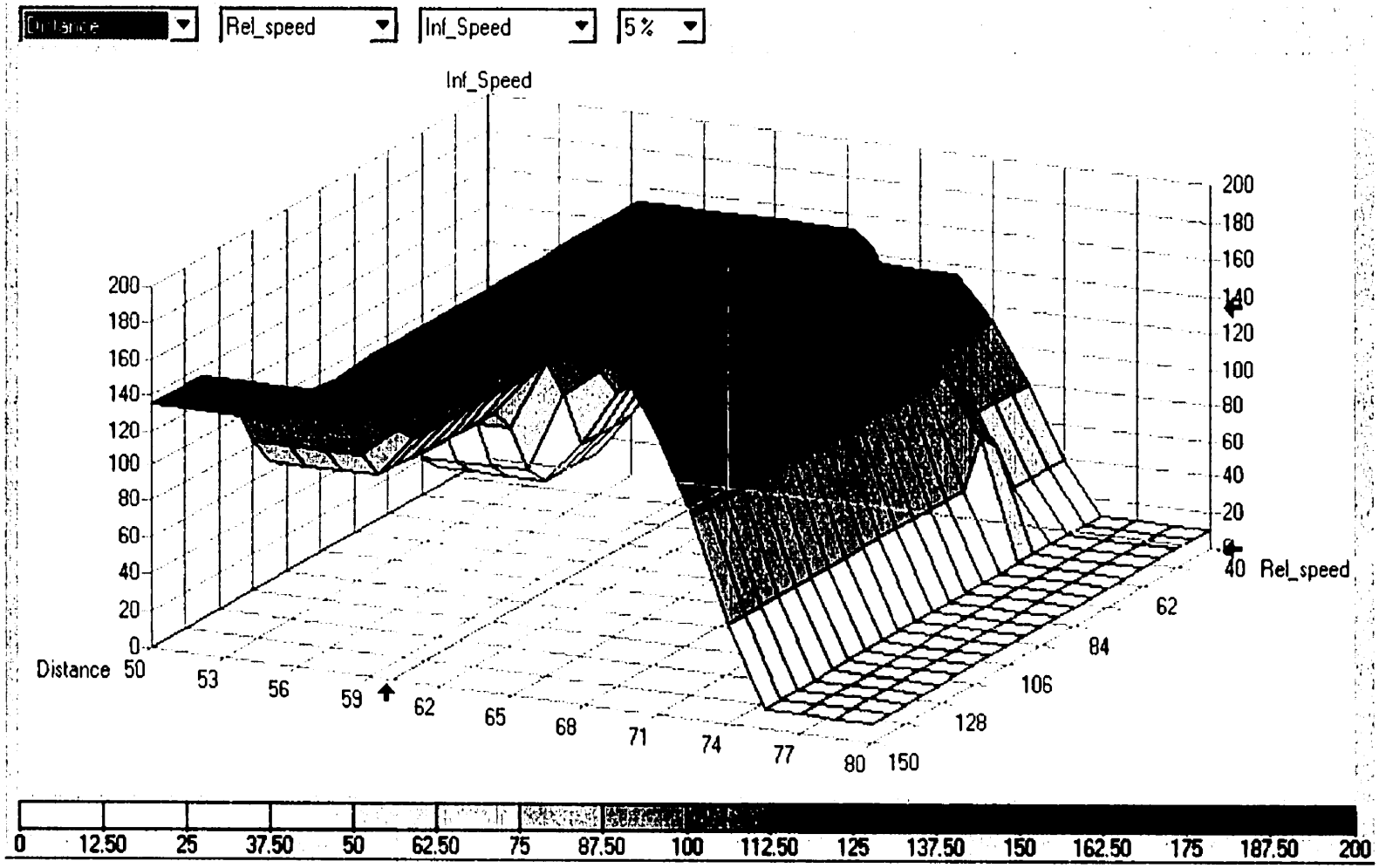

Fig: 32 "Inf _Speed" w.r.t. "Distance" and " Rel_Speed"

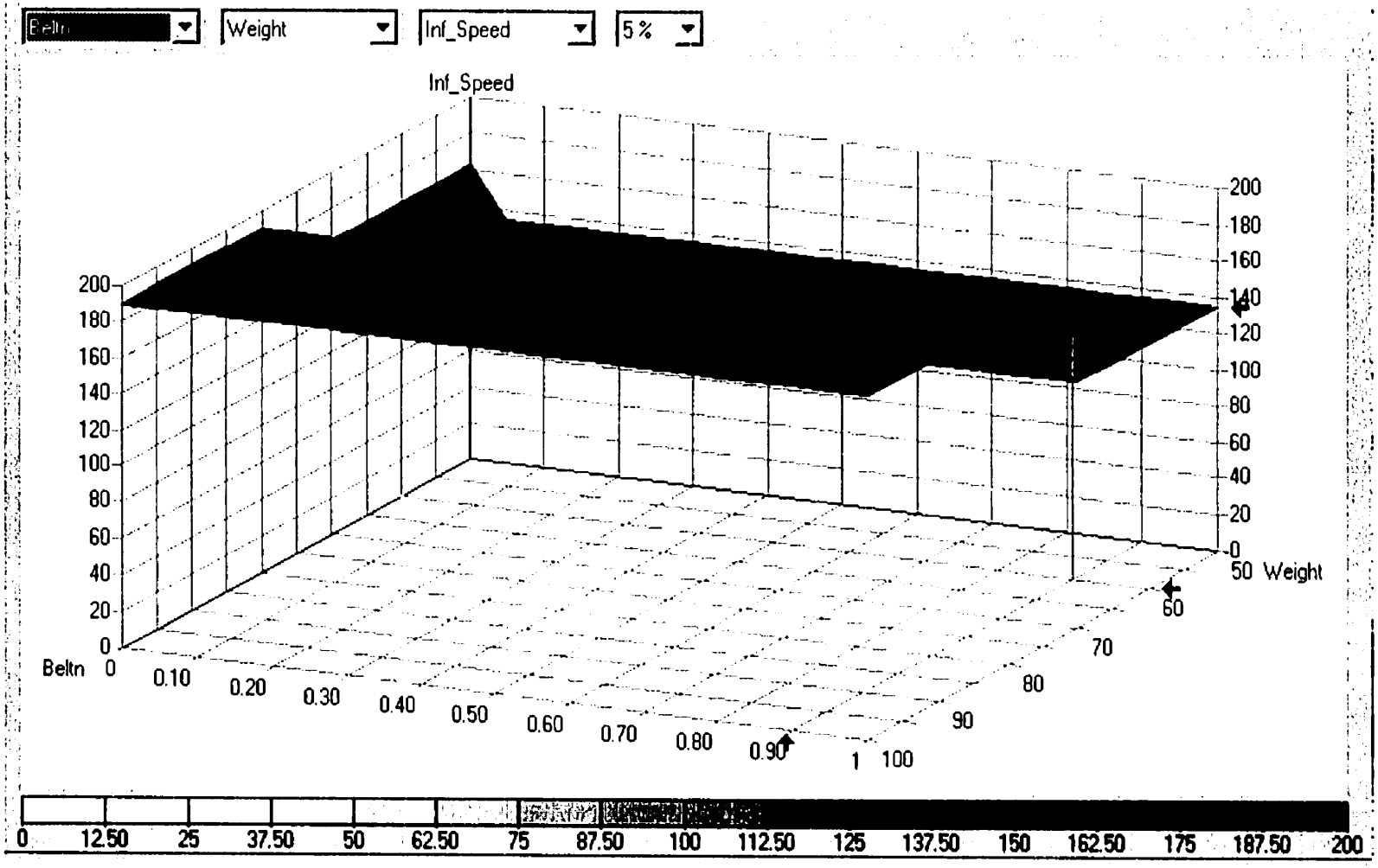

Fig: 33 "Inf_Speed" w.r.t. "Bltn" and "Weight

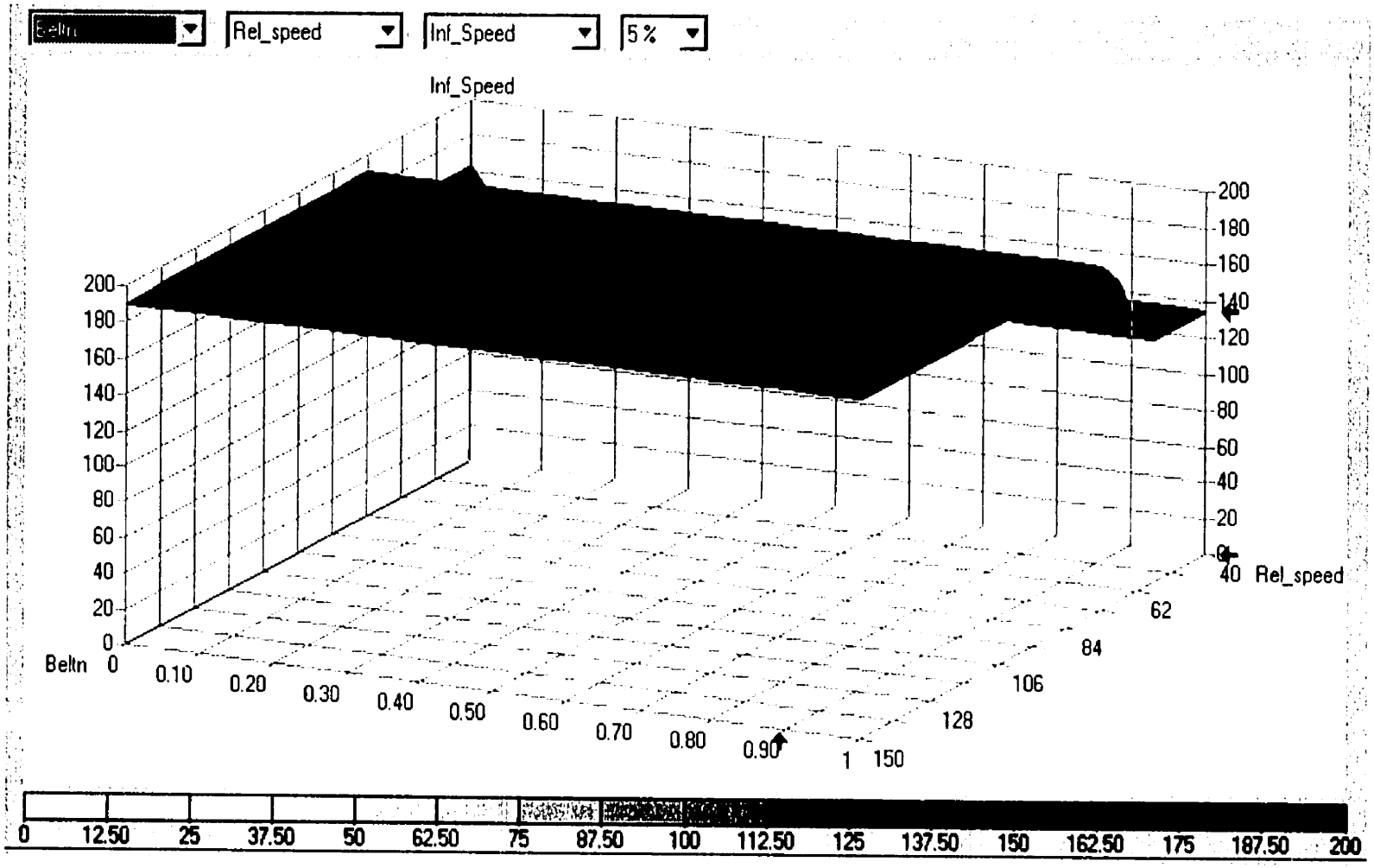

Fig: 34 "Inf_Speed" w.r.t. "Bltn" and "Rel_Speed"

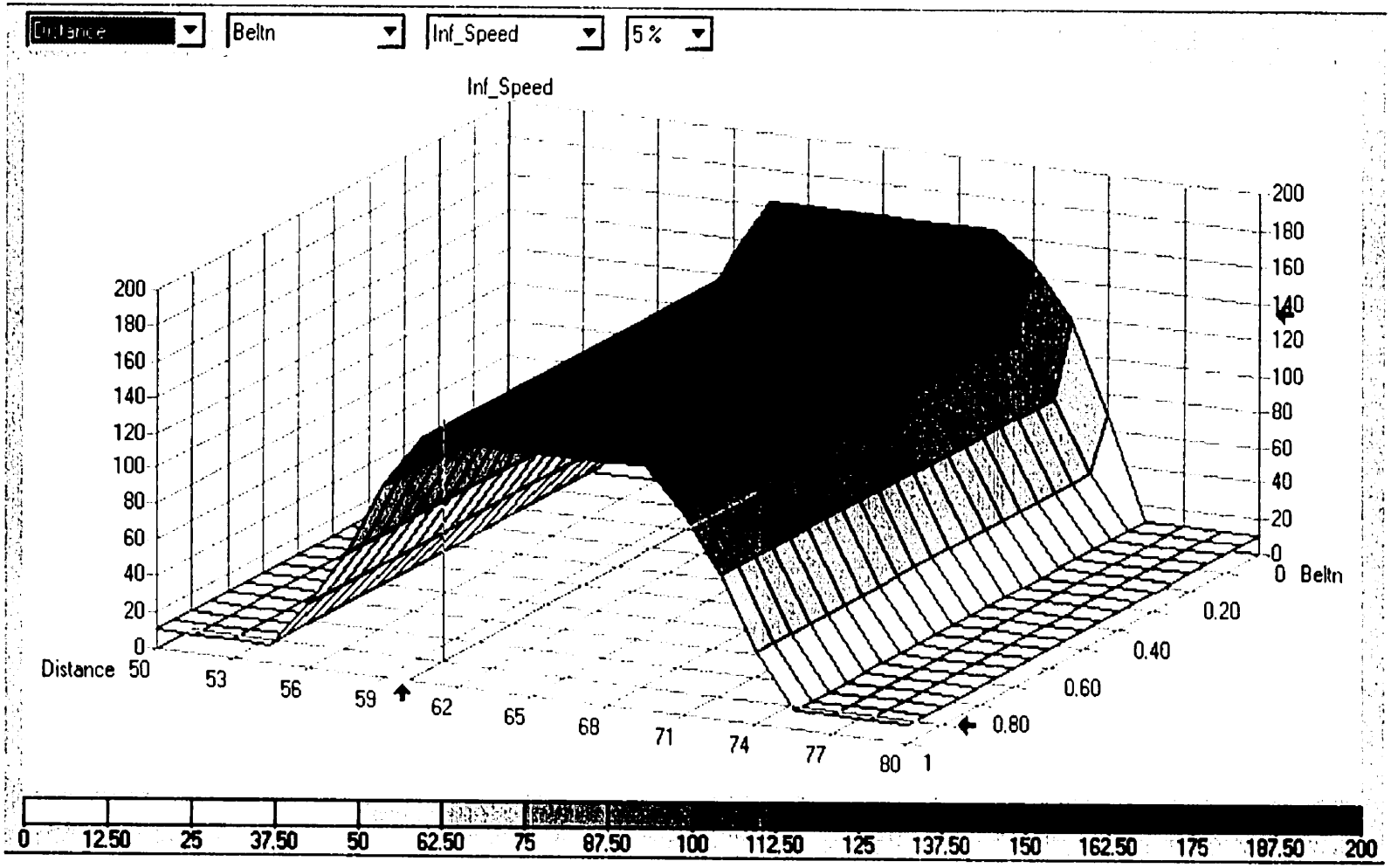

Fig: 35 "Inf_Speed" w.r.t. "Distance" and "Bltn"

CHAPTER 8

SUMMERY AND CONCLUSION

This thesis is concerned with the design techniques for fuzzy logic and its implementation in the automotive airbag control system. It should be noted that the techniques used here lend themselves to the flexibility of designing virtually any type of decision structure based on human linguistic variables. In chapter 1, we discussed the mechanism of airbag and also discussed the areas where they need improvement. In chapter 3 we discussed fuzzy logic concepts and decision structure bases for conclusion, reasoning and mathematical models for each technique. Due to the new series of microcontrollers 68HC12, which has dedicated instructions for programming and implementation of fuzzy logic it has become easy to write a smaller code which can overcome the memory constraints of earlier versions of microcontrollers. Automotive industry had been using 68HC11 for long time and it could handle fuzzy logic techniques for embedded systems too but the code for 68HC12 is one fifth in size and about fifteen times faster than its predecessors. Chapter 4 has an overview of the 68HC12, its memory structure, addressing modes and basic features including key improvements over 68HC11. Chapter 5 further discusses the fuzzy logic concepts in more detail and we saw that how an element belongs to the set with a degree of membership between 0 and 1. We showed that a fuzzy controller consists of three parts, input section, inference section and defuzzification section in which a fuzzy output is converted to a crisp output. In chapter 6, we have discussed the instruction set for fuzzy logic in detail. We showed how MEM

instruction can be used to perform the mapping of inputs to fuzzy set, rule evaluation by REV and REVW instructions and WAV instruction for defuzzification. The FuzzyTECH design software has made it very easy to design a control system using fuzzy logic. The reference manual for the microcontroller also advises to use some design tools for implementation and after trying different other tools, we realized its importance and preferred to use it. To ensure the stability and time response of the system, we generated a pattern file and observed its response on the 3-D graphs with different combinations of inputs. The output graph clearly shows different output levels and calculates defuzzified output result at the bottom.

This design approach using fuzzy logic is practically feasible and many other applications are open venues for further research and future work. There is a potential for future work for an ASIC, FPGA or custom chip design to perform these tasks.

REFERENCES

[1] Airbag - Supplemental Restraint Systems.

http://www.lemurzone.com/airbag/

[2] Automotive Safety.

http://www.autoliv.com/appl_alv/Autoliv.nsf/pages/front

[3] Safety Issues for Canadians.

http://www.tc.gc.ca/roadsafety/absg/airboce.htm

[4] Siemens Automotive News Release,

http://media.siemensauto.com/mediacenter2/queries/releasefull.phtml?prjob_num=1174

[5] Motorola, "68HC12 Referennce Manual," CPU12RM/AD Rev. pp 1.1- 1.4, 1997.

[6] Constantin Von Altrock, "Fuzzy Logic & NeuroFuzzy Applications Explained" pp 3—18, pp8—10, pp243—245, 1995.

[7] Tong, R. M., "Analysis of Fuzzy Control Algorithms Using the Relation Matrix," International Journal of Man-Machines Studies, pp. 679—686, 1976

[8] Zimmermann, H.-J. and Thole, U., "On the Suitability of Minimum and Product Operators for the Intersection of Fuzzy Sets," Fuzzy Sets and Systems, 2, pp. 173-186.

[9] Yamakawa, T., Shirai, Y. and Ueno, F., "Implementation of Fuzzy Logic Hardware Systems," *IEEE Transactionse*, Vol. C-63 (1980), pp. 720—725 and pp. 86 1—862

[10] L.A. Zadeh. Fuzzy sets. "Information and Control", pp. 338-383; 1965.

[11] M. Cayrol, H. Farency and H. Prade. "Fuzzy pattern matching", pp. 103--106; 1982.

[12] M. Mizumoto, S. Fukami, and K. Tanaka. Some Methods of Fuzzy Reasoning. In Advances in Fuzzy Set Theory and Applications, M.M. Gupta, R.K. Ragade, and R.R. Yager, eds. North-Holland, Amsterdam. 1979.pp.117-136.

[13] L.A. Zadeh. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. New York. 1973.

[14] Tzi-cker Chiueh. Optimization of fuzzy logic inference architecture. Computer, May:67-71; 1992.

[15] T. Whalen and B. Schott. Issues in fuzzy production systems. International Journal of Man-Machine Studies,19:57; 1983.

[16] Richard E. Haskell, "Design of Embedded System Using 68HC12/11 Microcontrollers", pp 361—385, 1999.

[17] James M. Sibigtroth, "Introducing, MC68HC12" http://www.mcu.motsps.com/

[18] User's Manual, FuzzyTECH 5.3

[19] Berkan, C. Riza and Trubatch, Sheldon L., "Fuzzy System Design Principles, Building Fuzzy IF—Then Rule Base", pp. 201—450, 1996.

[20] Schartz, D.G. & Klir, G. J. "Fuzzy Logic Flowers in Japan" IEEE Spectrum, pp. 32-35, July 1992

[21] William, T., "Alliance to Speed Acceptance of Fuzzy Logic Technology-Intell and Inform Software Team Up for 16-bit MCUs," Computer Design 12 (1992), pp 52—56

[22] Zimmermann, H.-J., Fuzzy Sets, Decision Making, and Expert Systems (1987), Boston, Dordrecht, London, ISBN 0-89838-149-5.

[23] Constantin Von Altrock, "Fuzzy Logic & NeuroFuzzy Applications Explained" pp 52—53, 1995.

[24] Bennett, J. Claude & Plum, Fred, "Cecil Text Book of Medecine", Vol 2, 20th 1152—1153, 1161—1163, 1994

[25] Mahan, L. Katheleen and Arlin, Marin T., "Krause's Food Nutrition & Diet Therapy", ed. 8th , 316—317, 1991

[26] Zimmerman H.-J. and Von Altrock, c. "Fuzzy Logic –Band 2", 1993

[27] "Europe Gets into Fuzzy Logic" (Electronics Engineering Times, Nov. 11, 1991).

[28] Emily T. Smith, "Why the Japanese are Going in for this 'Fuzzy Logic'" (Business Week, Feb. 20, 1993, pp. 39).

[29] Cox, E., "Fuzzy Fundamentals" (IEEE Spectrum, October 1992, pp. 58-61).

[30] Lee, C.C., "Fuzzy Logic in Control Systems" IEEE Trans. on Systems, Man, and Cybernetics, SMC, Vol. 20, No. 2, , pp. 404-35, 1990.

[31] Peterson, Ivars "Fuzzy Sets" Science News, Vol. 144, , pp. 55. July 24, 1993

[32] Sugeno, M. "Industrial Applications of Fuzzy Control" . North-Holland, New York, 1985.

[33] Jamshidi, Mohammad. " Fuzzy Logic and Control" environmental & Intelligent Series. pp. 95—110 1993

[34] Chen, C. H., "Fuzzy Logic and Neural Network Handbook" pp. 6.1—6.29, 1995

[35] "Fuzzy Logic - From Concept to Implementation", (Application Note EDU01V10-0193. (c) 1993 by Aptronix, Inc

[36] T. Whalen and B. Schott. "Issues in fuzzy production systems". International Journal of Man-Machine Studies,19--57; 1983.

[37] Kaufmann, A. and Gupta, M. M. "Fuzzy Mathematical Models in Engineering and Management Science".North-Holland. 1988.

[38] Jamshidi, M., Vadiee, N. & Ross, T., "Fuzzy Logic & Control: Software and Hardware Applications" Vol. 2, 1993

[39] Jamshide, M., Titli, A., Zadeh, L. and Boverie, S. " Applications of Fuzzy Logic: Towards High Machine Intelligence Quotient Systems", 1997

[40] An Overview of the 68HC12 http://www.seattlerobotics.org/encoder/jan97/The68HC12.html

[41] Sibigtroth, Jim." Agraphical Introduction to Fuzzy Logic", Motorola Microcontroller Technologies Group, Austin Texas, www.mcu.motsps.com/hc12/sib/intro1.html

[42] MC68HC912B32TS/D, Technical Summery 16_Bit Microcontroller

APPENDIX A

Design of Fuzzy Logic Linguistic Variables.

Computer Program

```
PROJECT {
 NAME      = AIRBAG.FTL;
 TITLE     =Airbag;
 AUTHOR    = Tariq M. Mian;
 DATEFORMAT = M.D.YYYY;
 LASTCHANGE = 12.10.1999;
 CREATED   = 11.18.1999;
 SHELL     = MCU_HC12;
 SHELLOPTIONS {
  ONLINE_REFRESHTIME  = 55;
  ONLINE_TIMEOUTCOUNT = 1100;
  ONLINE_CODE        = OFF;
  ONLINE_TRACE_BUFFER = (OFF, PAR(0));
  COMMENTS           = ON;
  FTL_BUFFER         = (OFF, PAR(1));
  PASSWORD           = OFF;
  PUBLIC_IO          = ON;
  FAST_CMBF          = OFF;
  FAST_COA           = OFF;
  FILE_CODE          = OFF;
  BTYPE              = 8_BIT;
  C_TYPE             = ANSI;
 } /* SHELLOPTIONS */
```

```
MODEL {
 VARIABLE_SECTION {
  LVAR {
   NAME      = Beltn;
   BASEVAR   = Units;
   LVRANGE   = MIN(0.000000), MAX(1.000000),
               MINDEF(0), MAXDEF(255),
               DEFAULT_OUTPUT(1.000000);
   RESOLUTION = XGRID(0.100000), YGRID(1.000000),
               SHOWGRID (ON), SNAPTOGRID(ON);
   COLOR     = RED (0), GREEN (128), BLUE (0);
   TERM {
    TERMNAME = Unbkl;
    POINTS   = (0.000000, 1.000000),
               (0.000000, 0.000000),
               (1.000000, 0.000000);
    SHAPE    = LINEAR;
    COLOR    = RED (255), GREEN (0), BLUE (0);
   }
   TERM {
    TERMNAME = Bkld;
    POINTS   = (0.000000, 1.000000),
               (0.600000, 1.000000),
```

```
        (1.000000, 1.000000);
  SHAPE   = LINEAR;
  COLOR   = RED (0), GREEN (0), BLUE (255);
 }
} /* LVAR */
LVAR {
 NAME     = Distance;
 BASEVAR  = Cm;
 LVRANGE  = MIN(50.000000), MAX(80.000000),
        MINDEF(0), MAXDEF(255),
        DEFAULT_OUTPUT(65.000000);
 RESOLUTION = XGRID(0.125000), YGRID(1.000000),
        SHOWGRID (ON), SNAPTOGRID(ON);
 COLOR    = RED (0), GREEN (128), BLUE (0);
 TERM {
  TERMNAME = Near;
  POINTS  = (50.000000, 1.000000),
        (55.000000, 1.000000),
        (60.000000, 0.000000),
        (80.000000, 0.000000);
  SHAPE   = LINEAR;
  COLOR   = RED (0), GREEN (128), BLUE (0);
 }
```

```
 TERM {
  TERMNAME = Far;
  POINTS   = (50.000000, 0.000000),
             (55.000000, 0.000000),
             (61.250000, 1.000000),
             (68.750000, 1.000000),
             (75.000000, 0.000000),
             (80.000000, 0.000000);
  SHAPE    = LINEAR;
  COLOR    = RED (0), GREEN (0), BLUE (255);
 }
 TERM {
  TERMNAME = Empty;
  POINTS   = (50.000000, 0.000000),
             (70.000000, 0.000000),
             (77.500000, 1.000000),
             (80.000000, 1.000000);
  SHAPE    = LINEAR;
  COLOR    = RED (128), GREEN (0), BLUE (0);
 }
} /* LVAR */
LVAR {
 NAME      = Rel_speed;
```

```
BASEVAR   = Kms;
LVRANGE   = MIN(40.000000), MAX(150.000000),
          MINDEF(0), MAXDEF(255),
          DEFAULT_OUTPUT(95.000000);
RESOLUTION = XGRID(0.500000), YGRID(1.000000),
          SHOWGRID (ON), SNAPTOGRID(ON);
COLOR     = RED (0), GREEN (0), BLUE (255);
TERM {
 TERMNAME = Cty;
 POINTS   = (40.000000, 1.000000),
          (50.000000, 1.000000),
          (70.000000, 0.000000),
          (150.000000, 0.000000) : OPEN (40.000000, 150.000000);
 SHAPE    = LINEAR;
 COLOR    = RED (255), GREEN (0), BLUE (0);
}
TERM {
 TERMNAME = Hiwy;
 POINTS   = (40.000000, 0.000000),
          (60.000000, 0.000000),
          (80.000000, 1.000000),
          (110.000000, 1.000000),
          (130.000000, 0.000000),
```

```
        (150.000000, 0.000000) : OPEN (40.000000, 150.000000);
  SHAPE   = LINEAR;
  COLOR   = RED (0), GREEN (128), BLUE (0);
 }
 TERM {
  TERMNAME = Ospd:
  POINTS  = (40.000000, 0.000000),
        (115.000000, 0.000000),
        (125.000000, 1.000000),
        (150.000000, 1.000000) : OPEN (40.000000, 150.000000);
  SHAPE   = LINEAR;
  COLOR   = RED (0), GREEN (0), BLUE (255);
 }
} /* LVAR */
LVAR {
 NAME     = Weight:
 BASEVAR   = Kgs;
 LVRANGE   = MIN(50.000000), MAX(100.000000),
        MINDEF(0), MAXDEF(255),
        DEFAULT_OUTPUT(75.000000);
 RESOLUTION = XGRID(0.200000), YGRID(1.000000),
        SHOWGRID (ON), SNAPTOGRID(ON);
 COLOR    = RED (255), GREEN (0), BLUE (0);
```

```
TERM {
 TERMNAME = Light;
 POINTS   = (50.000000, 1.000000),
          (52.600000, 1.000000),
          (60.600000, 0.000000),
          (100.000000, 0.000000);
 SHAPE    = LINEAR;
 COLOR    = RED (255), GREEN (0), BLUE (0);
}
TERM {
 TERMNAME = Bavg;
 POINTS   = (50.000000, 0.000000),
          (55.200000, 0.000000),
          (63.200000, 1.000000),
          (71.000000, 1.000000),
          (79.000000, 0.000000),
          (100.000000, 0.000000);
 SHAPE    = LINEAR;
 COLOR    = RED (0), GREEN (128), BLUE (0);
}
TERM {
 TERMNAME = Avg;
 POINTS   = (50.000000, 0.000000),
```

```
            (71.000000, 0.000000),
            (76.400000, 1.000000),
            (84.200000, 1.000000),
            (92.200000, 0.000000),
            (100.000000, 0.000000);
   SHAPE    = LINEAR;
   COLOR    = RED (0), GREEN (0), BLUE (255);
  }
  TERM {
   TERMNAME = Hvy;
   POINTS   = (50.000000, 0.000000),
            (84.200000, 0.000000),
            (92.200000, 1.000000),
            (100.000000, 1.000000);
   SHAPE    = LINEAR;
   COLOR    = RED (128), GREEN (0), BLUE (0);
  }
 } /* LVAR */
 LVAR {
  NAME      = Inf_Speed;
  BASEVAR   = Kmph;
  LVRANGE   = MIN(0.000000), MAX(200.000000),
            MINDEF(0), MAXDEF(255),
```

```
        DEFAULT_OUTPUT(100.000000);
RESOLUTION = XGRID(1.000000), YGRID(1.000000),
        SHOWGRID (ON), SNAPTOGRID(ON);
COLOR     = RED (128), GREEN (0), BLUE (0);
TERM {
 TERMNAME = Nil;
 POINTS   = (0.000000, 1.000000),
        (20.000000, 1.000000),
        (60.000000, 0.000000),
        (200.000000, 0.000000);
 SHAPE    = LINEAR;
 COLOR    = RED (255), GREEN (0), BLUE (0);
}
TERM {
 TERMNAME = Lvl1;
 POINTS   = (0.000000, 0.000000),
        (20.000000, 0.000000),
        (50.000000, 1.000000),
        (80.000000, 1.000000),
        (120.000000, 0.000000),
        (200.000000, 0.000000);
 SHAPE    = LINEAR;
 COLOR    = RED (0), GREEN (128), BLUE (0);
```

```
  }
  TERM {
   TERMNAME = Lvl2;
   POINTS   = (0.000000, 0.000000),
              (80.000000, 0.000000),
              (120.000000, 1.000000),
              (150.000000, 1.000000),
              (180.000000, 0.000000),
              (200.000000, 0.000000);
   SHAPE    = LINEAR;
   COLOR    = RED (128), GREEN (0), BLUE (0);
  }
  TERM {
   TERMNAME = Lvl3;
   POINTS   = (0.000000, 0.000000),
              (150.000000, 0.000000),
              (180.000000, 1.000000),
              (200.000000, 1.000000);
   SHAPE    = LINEAR;
   COLOR    = RED (128), GREEN (0), BLUE (0);
  }
 } /* LVAR */
} /* VARIABLE_SECTION */
```

```
OBJECT_SECTION {
 INTERFACE {
  INPUT = (Weight, CMBF);
  POS = -201, 135;
 }
 INTERFACE {
  INPUT = (Distance, CMBF);
  POS = -205, 10;
 }
 INTERFACE {
  INPUT = (Rel_speed, CMBF);
  POS = -203, 75;
 }
 INTERFACE {
  OUTPUT = (Inf_Speed, COM);
  POS = 196, 40;
 }
 INTERFACE {
  INPUT = (Beltn, CMBF);
  POS = -203, -58;
 }
 RULEBLOCK {
```

```
NAME       = RB1;
INPUT      = Beltn, Distance, Rel_speed, Weight;
OUTPUT     = Inf_Speed;
AGGREGATION = (MIN_MAX, PAR (0.000000));
RESULT_AGGR = MAX;
POS        = -18, 12;
RULES {
 IF   Beltn = Unbkl
  AND Distance = Near
  AND Rel_speed = Cty
  AND Weight = Light
 THEN  Inf_Speed = Lvl1   WITH 1.000;
 IF   Beltn = Unbkl
  AND Distance = Near
  AND Rel_speed = Cty
  AND Weight = Bavg
 THEN  Inf_Speed = Lvl1   WITH 1.000;
 IF   Beltn = Unbkl
  AND Distance = Near
  AND Rel_speed = Cty
  AND Weight = Avg
 THEN  Inf_Speed = Lvl2   WITH 1.000;
 IF   Beltn = Unbkl
```

AND Distance = Near

AND Rel_speed = Cty

AND Weight = Hvy

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Light

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Bavg

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Avg

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Hvy

```
THEN  Inf_Speed = Lvl2   WITH 1.000;

IF    Beltn = Unbkl

 AND Distance = Near

 AND Rel_speed = Ospd

 AND Weight = Light

THEN  Inf_Speed = Lvl2   WITH 1.000;

IF    Beltn = Unbkl

 AND Distance = Near

 AND Rel_speed = Ospd

 AND Weight = Bavg

THEN  Inf_Speed = Lvl3   WITH 1.000;

IF    Beltn = Unbkl

 AND Distance = Near

 AND Rel_speed = Ospd

 AND Weight = Avg

THEN  Inf_Speed = Lvl3   WITH 1.000;

IF    Beltn = Unbkl

 AND Distance = Near

 AND Rel_speed = Ospd

 AND Weight = Hvy

THEN  Inf_Speed = Lvl3   WITH 1.000;

IF    Beltn = Unbkl

 AND Distance = Far
```

AND Rel_speed = Cty

AND Weight = Light

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Cty

AND Weight = Bavg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Cty

AND Weight = Avg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Cty

AND Weight = Hvy

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Light

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Bavg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Avg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Hvy

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Light

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Bavg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Avg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Hvy

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Light

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Bavg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Avg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Hvy

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Hiwy

AND Weight = Light

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Hiwy

AND Weight = Bavg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Hiwy

AND Weight = Avg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Hiwy

AND Weight = Hvy

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Ospd

AND Weight = Light

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Ospd

AND Weight = Bavg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Ospd

AND Weight = Avg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Unbkl

AND Distance = Empty

AND Rel_speed = Ospd

AND Weight = Hvy

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Cty

AND Weight = Light

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Cty

AND Weight = Bavg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Cty

AND Weight = Avg

THEN Inf_Speed = Lvl1 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Cty

AND Weight = Hvy

THEN Inf_Speed = Lvl1 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Light

THEN Inf_Speed = Lvl1 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Bavg

THEN Inf_Speed = Lvl1 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Avg

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Hiwy

AND Weight = Hvy

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Ospd

AND Weight = Light

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Ospd

AND Weight = Bavg

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Ospd

AND Weight = Avg

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Near

AND Rel_speed = Ospd

AND Weight = Hvy

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Cty

AND Weight = Light

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Cty

AND Weight = Bavg

THEN Inf_Speed = Lvl2 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Cty

AND Weight = Avg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Cty

AND Weight = Hvy

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Light

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Bavg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Avg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Hiwy

AND Weight = Hvy

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Light

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Bavg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Avg

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Far

AND Rel_speed = Ospd

AND Weight = Hvy

THEN Inf_Speed = Lvl3 WITH 1.000;

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Light

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Bavg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Avg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Cty

AND Weight = Hvy

THEN Inf_Speed = Nil WITH 1.000:

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Hiwy

AND Weight = Light

THEN Inf_Speed = Nil WITH 1.000:

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Hiwy

AND Weight = Bavg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Hiwy

AND Weight = Avg

THEN Inf_Speed = Nil WITH 1.000;

IF Beltn = Bkld

AND Distance = Empty

AND Rel_speed = Hiwy

```
  AND Weight = Hvy
 THEN  Inf_Speed = Nil   WITH 1.000;
 IF    Beltn = Bkld
  AND Distance = Empty
  AND Rel_speed = Ospd
  AND Weight = Light
 THEN  Inf_Speed = Nil   WITH 1.000;
 IF    Beltn = Bkld
  AND Distance = Empty
  AND Rel_speed = Ospd
  AND Weight = Bavg
 THEN  Inf_Speed = Nil   WITH 1.000;
 IF    Beltn = Bkld
  AND Distance = Empty
  AND Rel_speed = Ospd
  AND Weight = Avg
 THEN  Inf_Speed = Nil   WITH 1.000;
 IF    Beltn = Bkld
  AND Distance = Empty
  AND Rel_speed = Ospd
  AND Weight = Hvy
 THEN  Inf_Speed = Nil   WITH 1.000;
} /* RULES */
```

```
} /* RULEBLOCK */
REMARK {
 TEXT = AUTOMOTIVE AIRBAG CONTROL SYSTEM;
 POS = -189, -170;
 FONTSPEC = -24, 700, 0, 0, 0, 34, 0;
 FONTNAME =Arial;
 COLOR = RED (0), GREEN (0), BLUE (0);
}
REMARK {
 TEXT = INPUTS;
 POS = -194, -113;
 FONTSPEC = -19, 400, 0, 0, 0, 34, 0;
 FONTNAME =Arial;
 COLOR = RED (0), GREEN (0), BLUE (0);
}
REMARK {
 TEXT = RULES BLOCK;
 POS = -26, -111;
 FONTSPEC = -19, 400, 0, 0, 0, 34, 0;
 FONTNAME =Arial;
 COLOR = RED (0), GREEN (0), BLUE (0);
}
REMARK {
```

```
      TEXT = OUTPUT;
      POS = 208, -110;
      FONTSPEC = -19, 400, 0, 0, 0, 34, 0;
      FONTNAME =Arial;
      COLOR = RED (0), GREEN (0), BLUE (0);
     }
   } /* OBJECT_SECTION */
  } /* MODEL */
} /* PROJECT */
ONLINE {
 TIMESTAMP = 19991210085058UT;
} /* ONLINE */

NEUROFUZZY {
 LEARNRULE    =RandomMethod:
 STEPWIDTHDOS  = 0.101563:
 STEPWIDTHTERM = 1.000000:
 MAXDEVIATION  = (50.000000, 1.000000, 0.750000);
 AVGDEVIATION  = 0.100000;
 MAXSTEPS      = 100:
 NEURONS       = 1;
 DATASEQUENCE  = RANDOM;
 UPDATEDBGWIN  = OFF; } /* NEUROFUZZY */
```

Appendix B1

Code Generator: C Source Code

Header File

```
/*---------------------------------------------------------------------*/
/*------------------- fuzzyTECH 5.30 MCU-HC11/12 Edition ------------------*/
/*-------------------- License Number: FT IU 00061 01 HS ------------------*/
/*---------------------------------------------------------------------*/
/*---------------------- Code Generator: C Source Code --------------------*/
/*------------- Code Generation Date: Fri Dec 10 03:54:36 1999 -----------*/
/*----------------------- Fuzzy Logic System: AIRBAG ----------------------*/
/*---------------------------------------------------------------------*/
/*-------- (c) 1991-1999 INFORM GmbH, Pascalstr. 23, D-52076 Aachen -------*/
/*------ Inform Software Corp., 2001 Midwest Rd., Oak Brook, IL 60523 -----*/
/*---------------------------------------------------------------------*/



/*---------------------------------------------------------------------*/
/*------------------- export interface of project AIRBAG ------------------*/
/*---------------------------------------------------------------------*/



/*---------------------------------------------------------------------*/
/*------------------------------- typedefs ------------------------------*/
/*---------------------------------------------------------------------*/
#ifndef FUZZYDEFINED
/*----------- type of data for computation of fuzzy logic system ----------*/
```

```
typedef unsigned char FUZZY;
#define FUZZYDEFINED
#endif

#ifndef FLAGSDEFINED
/*---------------- type of return value of fuzzy controller ---------------*/
typedef unsigned char FLAGS;
#define FLAGSDEFINED
#endif

/*---------------------- data only used by fuzzyTECH ---------------------*/
extern FUZZY * const pcvairbag;

/*-------------------------------------------------------------------------*/
/*- use the following #defines to write the inputs to the fuzzy controller */
/*-------------------------------------------------------------------------*/
#define Beltn_airbag                    (*(pcvairbag+  0)) /* 0000H .. 00FFH */
#define Distance_airbag                 (*(pcvairbag+  1)) /* 0000H .. 00FFH */
#define Rel_speed_airbag                (*(pcvairbag+  2)) /* 0000H .. 00FFH */
#define Weight_airbag                   (*(pcvairbag+  3)) /* 0000H .. 00FFH */
```

```
/*---------------------------------------------------------------------------*/
/* use the following #defines to read the outputs from the fuzzy controller */
/*---------------------------------------------------------------------------*/
#define Inf_Speed_airbag                (*(pcvairbag+ 4)) /* 0000H .. 00FFH */



/*---------------------------------------------------------------------------*/
/*-------------------------- function prototypes ----------------------------*/
/*---------------------------------------------------------------------------*/

/*------- for starting up the generated fuzzy logic system, call once -----*/
void initairbag(void);

/*-------------- for calling the generated fuzzy logic system -------------*/
FLAGS airbag(void);
```

APPENDIX B2

ANSI C code for 68HC12

```
/*---------------------------------------------------------------------*/
/*------------------ fuzzyTECH 5.30 MCU-HC11/12 Edition ----------------*/
/*------------------- License Number: FT IU 00061 01 HS ----------------*/
/*---------------------------------------------------------------------*/
/*---------------------- Code Generator: C Source Code ------------------*/
/*------------- Code Generation Date: Fri Dec 10 03:54:36 1999 ----------*/
/*----------------------- Fuzzy Logic System: AIRBAG --------------------*/
/*---------------------------------------------------------------------*/
/*-------- (c) 1991-1999 INFORM GmbH, Pascalstr. 23, D-52076 Aachen -------*/
/*------ Inform Software Corp., 2001 Midwest Rd., Oak Brook, IL 60523 -----*/
/*---------------------------------------------------------------------*/

#define FTLIBC8
#include "ftlibc.h"
#define FUZZYDEFINED
#define FLAGSDEFINED
#include "airbag.h"

static FUZZY crispio[4+1];

static FUZZY fuzvals[12+4+0];

FUZZY * const pcvairbag = crispio;
```

```
static const FUZZY tpts[48] = {
 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0xFF, 0xFF,
 0x00, 0x00, 0x2B, 0x55,
 0x2B, 0x60, 0x9F, 0xD5,
 0xAA, 0xEA, 0xFF, 0xFF,
 0x00, 0x00, 0x17, 0x46,
 0x2E, 0x5D, 0xA2, 0xD1,
 0xAE, 0xC5, 0xFF, 0xFF,
 0x00, 0x00, 0x0D, 0x36,
 0x1B, 0x43, 0x6B, 0x94,
 0x6B, 0x87, 0xAE, 0xD7,
 0xAE, 0xD7, 0xFF, 0xFF};

static const FUZZY xcom[4] = {
 0x0D, 0x53, 0xAC, 0xF2};

static const BYTE rt0[434] = {
 0x48, 0x00,
 0x03, 0x01, 0x02, 0x05, 0x08, 0x0D,
 0x03, 0x01, 0x02, 0x05, 0x09, 0x0D,
 0x03, 0x01, 0x02, 0x05, 0x0A, 0x0E,
```

0x03, 0x01, 0x02, 0x05, 0x0B, 0x0E,

0x03, 0x01, 0x02, 0x06, 0x08, 0x0E,

0x03, 0x01, 0x02, 0x06, 0x09, 0x0E,

0x03, 0x01, 0x02, 0x06, 0x0A, 0x0E,

0x03, 0x01, 0x02, 0x06, 0x0B, 0x0E,

0x03, 0x01, 0x02, 0x07, 0x08, 0x0E,

0x03, 0x01, 0x02, 0x07, 0x09, 0x0F,

0x03, 0x01, 0x02, 0x07, 0x0A, 0x0F,

0x03, 0x01, 0x02, 0x07, 0x0B, 0x0F,

0x03, 0x01, 0x03, 0x05, 0x08, 0x0F,

0x03, 0x01, 0x03, 0x05, 0x09, 0x0F,

0x03, 0x01, 0x03, 0x05, 0x0A, 0x0F,

0x03, 0x01, 0x03, 0x05, 0x0B, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x08, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x09, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x0A, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x0B, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x08, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x09, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x0A, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x0B, 0x0F,

0x03, 0x01, 0x04, 0x05, 0x08, 0x0C,

0x03, 0x01, 0x04, 0x05, 0x09, 0x0C,

0x03, 0x01, 0x04, 0x05, 0x0A, 0x0C,

0x03, 0x01, 0x04, 0x05, 0x0B, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x08, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x09, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x0A, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x0B, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x08, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x09, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x0A, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x0B, 0x0C,

0x03, 0x01, 0x02, 0x05, 0x08, 0x0C,

0x03, 0x01, 0x02, 0x05, 0x09, 0x0C,

0x03, 0x01, 0x02, 0x05, 0x0A, 0x0D,

0x03, 0x01, 0x02, 0x05, 0x0B, 0x0D,

0x03, 0x01, 0x02, 0x06, 0x08, 0x0D,

0x03, 0x01, 0x02, 0x06, 0x09, 0x0D,

0x03, 0x01, 0x02, 0x06, 0x0A, 0x0E,

0x03, 0x01, 0x02, 0x06, 0x0B, 0x0E,

0x03, 0x01, 0x02, 0x07, 0x08, 0x0E,

0x03, 0x01, 0x02, 0x07, 0x09, 0x0E,

0x03, 0x01, 0x02, 0x07, 0x0A, 0x0E,

0x03, 0x01, 0x02, 0x07, 0x0B, 0x0E,

0x03, 0x01, 0x03, 0x05, 0x08, 0x0E,

0x03, 0x01, 0x03, 0x05, 0x09, 0x0E,

0x03, 0x01, 0x03, 0x05, 0x0A, 0x0F,

0x03, 0x01, 0x03, 0x05, 0x0B, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x08, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x09, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x0A, 0x0F,

0x03, 0x01, 0x03, 0x06, 0x0B, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x08, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x09, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x0A, 0x0F,

0x03, 0x01, 0x03, 0x07, 0x0B, 0x0F,

0x03, 0x01, 0x04, 0x05, 0x08, 0x0C,

0x03, 0x01, 0x04, 0x05, 0x09, 0x0C,

0x03, 0x01, 0x04, 0x05, 0x0A, 0x0C,

0x03, 0x01, 0x04, 0x05, 0x0B, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x08, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x09, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x0A, 0x0C,

0x03, 0x01, 0x04, 0x06, 0x0B, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x08, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x09, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x0A, 0x0C,

0x03, 0x01, 0x04, 0x07, 0x0B, 0x0C};

```
static const FRAT frat0[3] = {
 0x0002, 0x00D8, 0x00D8};

FLAGS airbag(void) {
 fuzptr = (PFUZZY) fuzvals;
 tpptr  = (PFUZZY) tpts;

 crisp = crispio[0];
 bTNum = 2;
 flms();

 crisp = crispio[1];
 bTNum = 3;
 flms();

 crisp = crispio[2];
 bTNum = 3;
 flms();

 crisp = crispio[3];
 bTNum = 4;
 flms();
```

```
 pfuzvals = (PFUZZY) fuzvals;
 rtptr    = (PFTBYTE) rt0;
 fuzptr   = (PFUZZY) &fuzvals[0];
 fratptr  = (PFRAT)  frat0;
 bTNum    = 2;
 Min(); /* min aggregation */

 invalidflags = 0;
 fuzptr       = &fuzvals[12];
 xcomptr      = (PFUZZY) xcom;

 crispio[4] = 0x80;
 bTNum  = 4;
 defuzz = &crispio[4];
 com();

 return invalidflags;
}

void initairbag(void) {
 for (fuzptr = &fuzvals[12];
     fuzptr < &fuzvals[16];
```

```
    *fuzptr++ = 0);
}

/*
|-------------------------------------------------|
| Memory          |    RAM    |    ROM    |
|-------------------------------------------------|
| Fuzzy Logic System |  21 (0015H) |  494 (01EEH) |
|-------------------------------------------------|
| Total           |  21 (0015H) |  494 (01EEH) |
|-------------------------------------------------|
*/
```

APPENDIX C1

Code for HIWARE for 68HC12

Header File

```
/*---------------------------------------------------------------------*/
/*------------------- fuzzyTECH 5.30 MCU-HC11/12 Edition ------------------*/
/*-------------------- License Number: FT IU 00061 01 HS ------------------*/
/*---------------------------------------------------------------------*/
/*---------------------- Code Generator: C Source Code --------------------*/
/*------------- Code Generation Date: Fri Dec 10 03:55:25 1999 ------------*/
/*----------------------- Fuzzy Logic System: AIRBAG ----------------------*/
/*---------------------------------------------------------------------*/
/*-------- (c) 1991-1999 INFORM GmbH, Pascalstr. 23, D-52076 Aachen -------*/
/*------ Inform Software Corp., 2001 Midwest Rd., Oak Brook, IL 60523 -----*/
/*---------------------------------------------------------------------*/


/*---------------------------------------------------------------------*/
/*------------------- export interface of project AIRBAG ------------------*/
/*---------------------------------------------------------------------*/


/*---------------------------------------------------------------------*/
/*------------------------------- typedefs ----------------------------*/
/*---------------------------------------------------------------------*/
#ifndef FUZZYDEFINED
/*----------- type of data for computation of fuzzy logic system ----------*/
```

```
typedef unsigned char FUZZY;
#define FUZZYDEFINED
#endif

#ifndef FLAGSDEFINED
/*---------- type of return value of fuzzy controller ----------*/
typedef unsigned char FLAGS;
#define FLAGSDEFINED
#endif

/*-------------------------------------------------------------------------*/
/*---------------------------- Toolkit: HIWARE ----------------------------*/
/*-------------------------------------------------------------------------*/


extern FLAGS _invalidflags;

extern FUZZY _Beltn_airbag;                    /* 0000H .. 00FFH */
extern FUZZY _Distance_airbag;                 /* 0000H .. 00FFH */
extern FUZZY _Rel_speed_airbag;                /* 0000H .. 00FFH */
extern FUZZY _Weight_airbag;                   /* 0000H .. 00FFH */
extern FUZZY _Inf_Speed_airbag;                /* 0000H .. 00FFH */
```

```
/*---------------------------------------------------------------------------*/
/*- use the following #defines to write the inputs to the fuzzy controller */
/*---------------------------------------------------------------------------*/
#define Beltn_airbag                    _Beltn_airbag
#define Distance_airbag                 _Distance_airbag
#define Rel_speed_airbag                _Rel_speed_airbag
#define Weight_airbag                   _Weight_airbag


/*---------------------------------------------------------------------------*/
/* use the following #defines to read the outputs from the fuzzy controller */
/*---------------------------------------------------------------------------*/
#define Inf_Speed_airbag                _Inf_Speed_airbag

/*---------------------------------------------------------------------------*/
/*-------------------------- function prototypes ----------------------------*/
/*---------------------------------------------------------------------------*/

/*------- for starting up the generated fuzzy logic system, call once -----*/
void _initairbag(void);
#define initairbag _initairbag
```

```
/*---------- for calling the generated fuzzy logic system ----------*/

FLAGS _airbag(void);

#define airbag _airbag
```

APPENDIX C2

Code for HIWARE for 68HC12

```
;--------------------------------------------------------------------------
;--------------------- fuzzyTECH 5.30 MCU-HC11/12 Edition --------------------
;--------------------- License Number: FT IU 00061 01 HS ---------------------
;--------------------------------------------------------------------------
;------------------- Code Generator: Assembler Source Code -------------------
;--------------- Code Generation Date: Fri Dec 10 03:55:25 1999 --------------
;------------------------ Fuzzy Logic System: AIRBAG ------------------------
;--------------------------------------------------------------------------
;---------- (c) 1991-1999 INFORM GmbH, Pascalstr. 23, D-52076 Aachen ---------
;-------- Inform Software Corp., 2001 Midwest Rd., Oak Brook, IL 60523 -------
;--------------------------------------------------------------------------


;----------------------------- Toolkit: HIWARE ----------------------------


;---------------------------------- MCU: 12 ---------------------------------
            XDEF    _airbag
            XDEF    _initairbag


;------------------- input/output interface of controller -------------------
            XDEF    _Beltn_airbag           ;00H .. FFH
            XDEF    _Distance_airbag        ;00H .. FFH
            XDEF    _Rel_speed_airbag       ;00H .. FFH
            XDEF    _Weight_airbag          ;00H .. FFH
```

```
        XDEF    _Inf_Speed_airbag       ;00H .. FFH


;-------------------- external functions of fuzzy library ---------------------

        XREF    flms

        XREF    com



;-------------------- external variables of fuzzy library ---------------------

        XREF    fuzptr

        XREF    _invalidflags

        XREF    itcnt

        XREF    tpptr

        XREF    crisp

        XREF    otcnt

;------------------------------------------------------------------------------

 ftData: SECTION 1


;------------------------------ RAM i/o-vars ----------------------------------

fuzvals:                                ;12 + 4 + 0

fvs:

_t_Beltn_airbag:    ds.b    2

_t_Distance_airbag: ds.b    3

_t_Rel_speed_airbag:    ds.b    3
```

```
_t_Weight_airbag:  ds.b  4
_t_Inf_Speed_airbag:  ds.b  4

crispio:                    :5
_Beltn_airbag:    ds.b  1
_Distance_airbag:  ds.b  1
_Rel_speed_airbag:  ds.b  1
_Weight_airbag:    ds.b  1
_Inf_Speed_airbag:  ds.b  1

ftCode: SECTION 2
;---------- standard term definition (x1, x2, x3, x4) ----------
tpts:
        dc.b  $00, $00, $00, $00
        dc.b  $00, $00, $FF, $FF
        dc.b  $00, $00, $2B, $55
        dc.b  $2B, $60, $9F, $D5
        dc.b  $AA, $EA, $FF, $FF
        dc.b  $00, $00, $17, $46
        dc.b  $2E, $5D, $A2, $D1
        dc.b  $AE, $C5, $FF, $FF
        dc.b  $00, $00, $0D, $36
        dc.b  $1B, $43, $6B, $94
```

```
        dc.b    $6B, $87, $AE, $D7

        dc.b    $AE, $D7, $FF, $FF


;------------------------ xcom table (defuzzification) ----------------------

xcom:

        dc.b    $0D, $53, $AC, $F2


;------------------------------ rule table(s) ------------------------------

_rt0:

        dc.b    $0, $2, $5, $8, $FE

        dc.b    $D, $FE

        dc.b    $0, $2, $5, $9, $FE

        dc.b    $D, $FE

        dc.b    $0, $2, $5, $A, $FE

        dc.b    $E, $FE

        dc.b    $0, $2, $5, $B, $FE

        dc.b    $E, $FE

        dc.b    $0, $2, $6, $8, $FE

        dc.b    $E, $FE

        dc.b    $0, $2, $6, $9, $FE

        dc.b    $E, $FE

        dc.b    $0, $2, $6, $A, $FE

        dc.b    $E, $FE
```

```
dc.b    $0, $2, $6, $B, $FE
dc.b    $E, $FE
dc.b    $0, $2, $7, $8, $FE
dc.b    $E, $FE
dc.b    $0, $2, $7, $9, $FE
dc.b    $F, $FE
dc.b    $0, $2, $7, $A, $FE
dc.b    $F, $FE
dc.b    $0, $2, $7, $B, $FE
dc.b    $F, $FE
dc.b    $0, $3, $5, $8, $FE
dc.b    $F, $FE
dc.b    $0, $3, $5, $9, $FE
dc.b    $F, $FE
dc.b    $0, $3, $5, $A, $FE
dc.b    $F, $FE
dc.b    $0, $3, $5, $B, $FE
dc.b    $F, $FE
dc.b    $0, $3, $6, $8, $FE
dc.b    $F, $FE
dc.b    $0, $3, $6, $9, $FE
dc.b    $F, $FE
dc.b    $0, $3, $6, $A, $FE
```

```
dc.b    $F, $FE
dc.b    $0, $3, $6, $B, $FE
dc.b    $F, $FE
dc.b    $0, $3, $7, $8, $FE
dc.b    $F, $FE
dc.b    $0, $3, $7, $9, $FE
dc.b    $F, $FE
dc.b    $0, $3, $7, $A, $FE
dc.b    $F, $FE
dc.b    $0, $3, $7, $B, $FE
dc.b    $F, $FE
dc.b    $0, $4, $5, $8, $FE
dc.b    $C, $FE
dc.b    $0, $4, $5, $9, $FE
dc.b    $C, $FE
dc.b    $0, $4, $5, $A, $FE
dc.b    $C, $FE
dc.b    $0, $4, $5, $B, $FE
dc.b    $C, $FE
dc.b    $0, $4, $6, $8, $FE
dc.b    $C, $FE
dc.b    $0, $4, $6, $9, $FE
dc.b    $C, $FE
```

```
dc.b    $0, $4, $6, $A, $FE
dc.b    $C, $FE
dc.b    $0, $4, $6, $B, $FE
dc.b    $C, $FE
dc.b    $0, $4, $7, $8, $FE
dc.b    $C, $FE
dc.b    $0, $4, $7, $9, $FE
dc.b    $C, $FE
dc.b    $0, $4, $7, $A, $FE
dc.b    $C, $FE
dc.b    $0, $4, $7, $B, $FE
dc.b    $C, $FE
dc.b    $1, $2, $5, $8, $FE
dc.b    $C, $FE
dc.b    $1, $2, $5, $9, $FE
dc.b    $C, $FE
dc.b    $1, $2, $5, $A, $FE
dc.b    $D, $FE
dc.b    $1, $2, $5, $B, $FE
dc.b    $D, $FE
dc.b    $1, $2, $6, $8, $FE
dc.b    $D, $FE
dc.b    $1, $2, $6, $9, $FE
```

```
dc.b    $D, $FE
dc.b    $1, $2, $6, $A, $FE
dc.b    $E, $FE
dc.b    $1, $2, $6, $B, $FE
dc.b    $E, $FE
dc.b    $1, $2, $7, $8, $FE
dc.b    $E, $FE
dc.b    $1, $2, $7, $9, $FE
dc.b    $E, $FE
dc.b    $1, $2, $7, $A, $FE
dc.b    $E, $FE
dc.b    $1, $2, $7, $B, $FE
dc.b    $E, $FE
dc.b    $1, $3, $5, $8, $FE
dc.b    $E, $FE
dc.b    $1, $3, $5, $9, $FE
dc.b    $E, $FE
dc.b    $1, $3, $5, $A, $FE
dc.b    $F, $FE
dc.b    $1, $3, $5, $B, $FE
dc.b    $F, $FE
dc.b    $1, $3, $6, $8, $FE
dc.b    $F, $FE
```

```
dc.b    $1, $3, $6, $9, $FE
dc.b    $F, $FE
dc.b    $1, $3, $6, $A, $FE
dc.b    $F, $FE
dc.b    $1, $3, $6, $B, $FE
dc.b    $F, $FE
dc.b    $1, $3, $7, $8, $FE
dc.b    $F, $FE
dc.b    $1, $3, $7, $9, $FE
dc.b    $F, $FE
dc.b    $1, $3, $7, $A, $FE
dc.b    $F, $FE
dc.b    $1, $3, $7, $B, $FE
dc.b    $F, $FE
dc.b    $1, $4, $5, $8, $FE
dc.b    $C, $FE
dc.b    $1, $4, $5, $9, $FE
dc.b    $C, $FE
dc.b    $1, $4, $5, $A, $FE
dc.b    $C, $FE
dc.b    $1, $4, $5, $B, $FE
dc.b    $C, $FE
dc.b    $1, $4, $6, $8, $FE
```

```
        dc.b    $C, $FE
        dc.b    $1, $4, $6, $9, $FE
        dc.b    $C, $FE
        dc.b    $1, $4, $6, $A, $FE
        dc.b    $C, $FE
        dc.b    $1, $4, $6, $B, $FE
        dc.b    $C, $FE
        dc.b    $1, $4, $7, $8, $FE
        dc.b    $C, $FE
        dc.b    $1, $4, $7, $9, $FE
        dc.b    $C, $FE
        dc.b    $1, $4, $7, $A, $FE
        dc.b    $C, $FE
        dc.b    $1, $4, $7, $B, $FE
        dc.b    $C, $FF

;-------------------------------------------------------------------------

_airbag:
;------------------------------ fuzzification ------------------------------
        ldy     #fuzvals
        ldd     #tpts
        std     tpptr
```

```
ldab   #$2
stab   itcnt
ldab   _Beltn_airbag
stab   crisp
jsr    flms

ldab   #$3
stab   itcnt
ldab   _Distance_airbag
stab   crisp
jsr    flms

ldab   #$3
stab   itcnt
ldab   _Rel_speed_airbag
stab   crisp
jsr    flms

ldab   #$4
stab   itcnt
ldab   _Weight_airbag
stab   crisp
jsr    flms
```

```
;------------------------------ inference init ------------------------------
;------------------------------ rule block 0 ------------------------------
            ldx    #_rt0
            ldy    #fuzvals
            ldaa   #255
            rev                         :min aggregation

;------------------------------ defuzzification ------------------------------
            clr    _invalidflags
            ldy    #fuzvals + SC

            ldx    #xcom + $0
            ldab   #$4
            stab   otcnt
            jsr    com
            bcc    out0valid
            ldab   #$80
out0valid:     stab   _Inf_Speed_airbag

            ldab   _invalidflags
            rts                         ;end of fuzzy controller
```

```
;-------------------------------- initairbag ------------------------------

_initairbag:
            ldy    #$4
            ldaa   #0
begin:
            staa   fuzvals + $B,y
            dey
            bne    begin
            rts

:data size knowledge base (bytes):
:RAM:       21   00015H
:ROM:      412   0019CH
:TOTAL:    433   001B1H
;
            end
```

VITA AUCTORIS

Tariq M. Mian

1964 Born on February 16th, Sheikhupura, Pakistan.

1978 Higher Secondary School, Applied Sciences,

Government High School, Sheikhupura, Pakistan.

1983 B. Sc. Mathematics and Physics,

University of the Punjab, Lahore, Pakistan.

1990 B. Sc Electrical Engineering,

University of Engineering & Technology, Lahore, Pakistan.

1999 Candidate for the degree of Master of Applied Science,

Electrical and Computer Engineering,

University of Windsor, Windsor, Ontario, Canada.