3-31-1992

Fuzzy special logic functions and applications

Te-Shun Chou
Florida International University

DOI: 10.25148/etd.FI14060817

ABSTRACT OF THE THESIS

Fuzzy Special Logic Functions and Applications

by

Te-Shun Chou

Florida International University, 1992

Miami, Florida

Professor Edward T. Lee, Major Professor

In this thesis, four special logic functions (threshold functions, monotone increasing functions, monotone decreasing functions, and unate functions) are extended to more general functions which allows the activities of these special functions to be a "fuzzy" rather than a "1-or-0" process. These special logic functions are called as fuzzy special logic functions and are based on the concepts and techniques developed in fuzzy logic and fuzzy languages. The algorithms of determining C(n), $C_{max}(n)$ and generating the most dissimilar fuzzy special logic functions as well as important properties and results are investigated. Examples are given to illustrated these special logic functions. In addition, their applications -- function representation, data compression, error correction, and monotone flash analog to digital converter, their relationships, and fuzzy classification are also presented. It is obviously shown that fuzzy logic theory can be used successfully on these four special logic functions in order to normalize the grade of membership function μ in the interval [0 1]. As a result, the techniques described in this thesis may be of use in the study of other special logic functions and much fertile field work is great worth researching and developing.

FLORIDA INTERNATIONAL UNIVERSITY
Miami, Florida

Fuzzy Special Logic Functions and Applications

A thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science
in Electrical and Computer Engineering

by

Te-Shun Chou

1992

To Professors: Dr. Wunnava V. Subbarao
Dr. Kang k. Yen
Dr. Edward T. Lee

This thesis, having been approved in respect to form and mechanical execution, is referred to you for judgement upon its substantial merit.

Dean, Gordon R. Hopkins
College of Engineering and Design

The thesis of Te-Shun Chou is approved.

Wunnava V. Subbarao

Kang k. Yen

Edward T. Lee, Major Professor

James R. Story, Chairman of Department of Electrical and Computer Engineering

Date of Examination: March 31, 1992

Dean, Richard Campbell
Division of Graduate Studies

Florida International University, 1992

To

my parents

ACKNOWLEDGEMENTS

I would like to acknowledge all who help in the completion of this thesis. First of all, I had to thank all my committee members for their thorough reading of my thesis as well as for their valuable advice and suggestions. Among them, I'd like to express my gratitude to Dr. Kang K. Yen. He is so influential not only in my study, but also in my life philosophy. Meanwhile, mostly I'd like to thank my major professor, Dr. Edward T. Lee, whose influence on me is profound. I was very fortunate to have the opportunity to learn with him. He provided me consistent help and advice, however the most important is that he taught me how to view a problem in a different way and researched it step by step. In a word, he is truly a good and helpful teacher.

In the meantime, I would be remiss if I did not mention my best friend, Mr. Ke-Wen Zeng. He is a considerate person who always tried to help me in every way. Thanks also to another good friend, Miss Ya-Fen Lee, who always supports and encourages me both in my personal life and the completion of this thesis.

Furthermost, I would like to give my deepest gratitude to all my families, especially my parents, P'eng-Hsiao Chou and Shiang-Lin W. Chou. They raised me with all their hearts. Without their love and support, I could never have achieved my master program. I want to share this wonderful honor with them. Again, thanks to them truly from the bottom of my heart.

Table of Contents

1. Fuzzy Threshold functions and Applications 1

1.1 Introduction 1

1.2 Threshold functions (Linearly separable functions) 2

1.2.1 Properties of threshold gates 4

1.2.2 Advantages of threshold logic 5

1.3 Fuzzy threshold functions 8

1.4 Determination of ρ_n 9

1.4.1 For two variables 9

1.4.2 For three variables 13

1.4.3 For n variables 16

1.4.4 Summary 17

1.5 An algorithm for determining $C_{max}(n)$ 17

1.6 An algorithm for determining C(n) 18

1.7 An algorithm for generating most inseparable functions 23

1.8 Applications 25

1.8.1 Function representation 25

1.8.2 Data compression 26

1.8.3 Error detection and correction 26

1.9 Conclusion 27

2. Fuzzy Monotone functions and Applications 28

2.1 Introduction 28

2.2 Monotone increasing functions 28

2.3 Fuzzy monotone increasing functions 29

2.4 Determination of ρ_n 30

2.4.1 For two variables 31

2.4.2 For three variables 32

2.4.3 For n variables 33

2.4.4 Summary 33

2.5 An algorithm for determining $C_{max}(n)$ 35

2.6 An algorithm for determining C(n) 36

2.7 An algorithm for generating most variation functions 39

2.8 Monotone decreasing functions 41

2.9 Fuzzy monotone decreasing functions 41

2.10 Applications 43

2.10.1 Function representation 43

2.10.2 Data compression 43

2.10.3 Error detection and correction 44

2.10.4 Analog to digital converters 45

2.11 Conclusion 45

3. Fuzzy Unate functions and Applications 46

3.1 Introduction 46

3.2 Unate functions 46

3.3 Fuzzy unate functions 47

3.4 Determination of ρ_n 48

3.4.1 For two variables 48

3.4.2 For three variables 50

3.4.3 For n variables 52

3.4.4 Summary 52

3.5 An algorithm for determining $C_{max}(n)$ 54

3.6 An algorithm for determining C(n) 56

3.7 An algorithm for generating most non-unate functions 59

3.8 Applications 60

3.8.1 Function representation 60

3.8.2 Data compression 61

3.8.3 Error detection and correction 61

3.9 Conclusion 62

4. Flash Analog to Digital Converters 63

4.1 Introduction 63

4.2 Basic structure 64

4.3 2-bit flash analog to digital converter 65

4.4 3-bit flash analog to digital converter 67

4.5 4-bit flash analog to digital converter 69

4.6 Conclusion 71

5. Minimum Universal Synthesis of Logic Functions by Using Monotone Flash Analog to Digital Converters 72

5.1 Introduction 72

5.2 Rule of choosing minterms or maxterms to synthesize a given function 72

5.3 Synthesis of a logic function by using monotone flash A/D converters 76

5.3.1 Synthesis of a logic function by using monotone increasing flash A/D converters 77

5.3.2 Synthesis of a logic function by using monotone decreasing flash A/D converters 83

5.4 Conclusion 87

6. Fuzzy Classification 88

6.1 Introduction 88

6.2 Generalization 89

6.2.1 Fuzzy threshold functions 90

6.2.2 Fuzzy monotone functions 91

6.2.2 Fuzzy unate functions 91

6.3 Relationships 93

6.4 Classification 96

6.5 Conclusion 99

7. Recommendations 100

References 101

Chapter One

Fuzzy Threshold Functions and Applications

1.1 Introduction

Digital computers, digital communication and control systems, and many other digital systems deal with two general problems: the characterization of the input-output relations of digital networks (using, for example, Boolean algebra), and the synthesis of digital networks from their input-output relations using certain specified logic elements as building blocks. Conventionally, these building blocks include such elements as AND, OR, and NOR gates.

All of these conventional elements share two very useful properties: they are easily constructed from standard components, and their input-output relations are simple and easily described in Boolean algebra. However, since new circuit techniques now promise the inexpensive mass fabrication of stable, reliable components, there is renewed interest in using a logically more powerful basic building block -- the threshold gate and McNaughton had the first published work in this field in 1957 [16]. Although this type of gate requires tighter tolerances than conventional gates, it promises a substantial reduction in the number of gates, the total number of components, and the number of interconnections.

Therefore, I would like to discuss this subject and use the concepts and techniques developed in fuzzy logic and fuzzy languages to fuzzify the kind of functions and their some applications in this chapter.

1.2 Threshold functions f_T (Linearly separable functions)

A threshold gate which is a logic device with n binary-valued inputs, $x_1, x_2, ..., x_n$, and a single binary-valued output y. Associated with each input x_i is a weight w_i. The values of the threshold T and w_i (i=1, 2,..., n) may be any real, finite, positive or negative numbers. The output y of the threshold gate is decided by as follows [9,15,16]:

$$y = 1 \quad \text{if and only if} \quad \sum_{i=1}^{n} w_i x_i \geq T \qquad [1.1]$$

$$y = 0 \quad \text{if and only if} \quad \sum_{i=1}^{n} w_i x_i < T \qquad [1.2]$$

where the sum and product operations are arithmetic. A threshold gate is represented pictorially as shown in Figure 1.1.

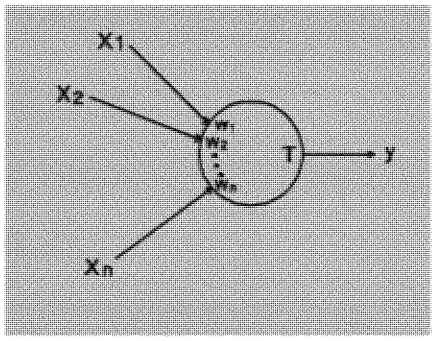


Figure 1.1. The symbol of a threshold gate

Clearly, a threshold gate realizes a Boolean function. Hence, the output of a threshold gate can be represented by a Boolean function as definition 1.1.

DEFINITION 1.1. [15,16] A threshold function is defined as:

$$y = \langle \sum_{i=1}^{n} w_i x_i \rangle_T \qquad [1.3]$$

where equation 1.3 is called the threshold function representation of the gate.

DEFINITION 1.2. A function is an inseparable function while it is not a threshold function.

Example 1.1. Consider the threshold gate as shown in Figure 1.2.

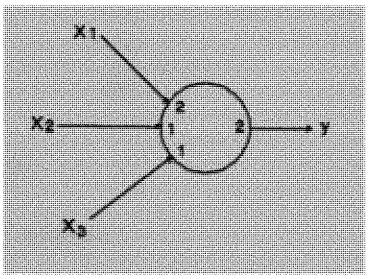


Figure 1.2 threshold gate that realizes $y = x_1+x_2x_3$

$$y = \langle 2x_1+x_2+x_3 \rangle_2$$

The Boolean function realized by the gate can be determined by constructing the truth table as shown in Table 1.1.

x_1,x_2,x_3	Threshold function $2x_1+x_2+x_3$	Relation to T=2	y
000	0	<	0
001	1	<	0
010	1	<	0
011	2	=	1
100	2	=	1
101	3	>	1
110	3	>	1
111	4	>	1

Table 1.1

From the truth table, we can get the following Boolean function which is realized by the threshold gate:

$$y = x_1+x_2x_3$$

1.2.1 Properties of threshold gates

There are some well-known properties of threshold gate as follows [9].

Let

$$y = \langle \sum_{i=1}^{n} w_i x_i \rangle_T \qquad [1.4]$$

be the output of a threshold gate.

(1) IF $w_1=w_2=...=w_n=T$, THEN

y is the Boolean function $x_1+x_2+...+x_n$.

(2) IF $w_1=w_2=...=w_n$ and $T=nw_1$, THEN

y is the Boolean function $x_1x_2...x_n$.

(3) IF k is a real number, THEN

$$y = \langle k+\sum_{i=1}^{n} w_i x_i \rangle_{k+T} \qquad [1.5]$$

(4) IF k is a positive real number, THEN

$$y = \langle k\sum_{i=1}^{n} w_i x_i \rangle_{kT} \qquad [1.6]$$

(5) IF k is a positive real number, THEN

$$y' = \langle -\sum_{i=1}^{n} w_i x_i \rangle_{-T+1} \qquad [1.7]$$

Example 1.2. Using property 1, a threshold gate realization of a three-variable OR is given below:

$$x_1+x_2+x_3 = \langle x_1+x_2+x_3 \rangle_1$$

Example 1.3. Using property 2, a threshold gate realization of a three-variable AND is as follows:

$$x_1x_2x_3 = \langle x_1+x_2+x_3 \rangle_3$$

Example 1.4. Property 3 is useful for changing the signs of coefficients.

$$y = \langle -2x_1-x_2+x_3 \rangle_{-1}$$

Determine an equivalent realization that only requires positive weights. That is $x_i=1-x_i'$. Then:

$$y = \langle -2(1-x_1')-(1-x_2')+x_3 \rangle_{-1}$$

$$= \langle -2+2x_1'-1+x_2'+x_3 \rangle_{-1}$$

$$= \langle -3+2x_1'+x_2'+x_3 \rangle_{-1}$$

$$= \langle 3-3+2x_1'-1+x_2'+x_3 \rangle_{3-1}$$

$$= \langle 2x_1'+x_2'+x_3 \rangle_2$$

1.2.2 Advantages of threshold logic

THEOREM 1.1. [9] A threshold gate for which all weights have unit value and the threshold value is 0.5 is a n-input OR gate.

Example 1.5. For a two-variable function which satisfies theorem 1.1.

x_1,x_2	x_1+x_2	$y = \langle x_1+x_2 \rangle_{0.5}$
00	0	0
01	1	1
10	1	1
11	1	1

After calculation, we have found this threshold gate is equal to a 2-input OR gate.

THEOREM 1.2. [9] A threshold gate for which all weights have unity value and the threshold value is n-0.5 is a n-input AND gate.

Example 1.6. For a two-variable function which satisfies theorem 1.2.

x_1,x_2	x_1x_2	$y = \langle x_1x_2 \rangle_{0.5}$
00	0	0
01	0	0
10	0	0
11	1	1

After calculation, we found this threshold gate is equal to a 2-input AND gate.

THEOREM 1.3. [9] In a similar situation, the complementing threshold gate is a generalization of the NOR and NAND gates.

Because of the wide range of possible weights and threshold values, the size and complexity of the class of logical functions that can be realized by a single threshold gate are large when compared with those of the conventional logical elements. As a result, the number of gates, inputs, and total number of physical components needed to realize a particular logical function are smaller for threshold gates than for conventional gates.

Example 1.7. Consider a seven-variable logic function

$$y=x_1(x_2+x_3+x_4+x_5+x_6+x_7)+x_2(x_3+x_4+x_5+x_6+x_7)+x_3(x_4+x_5+x_6+x_7)+x_4(x_5+x_6+x_7)+x_5(x_6+x_7)+x_6x_7$$

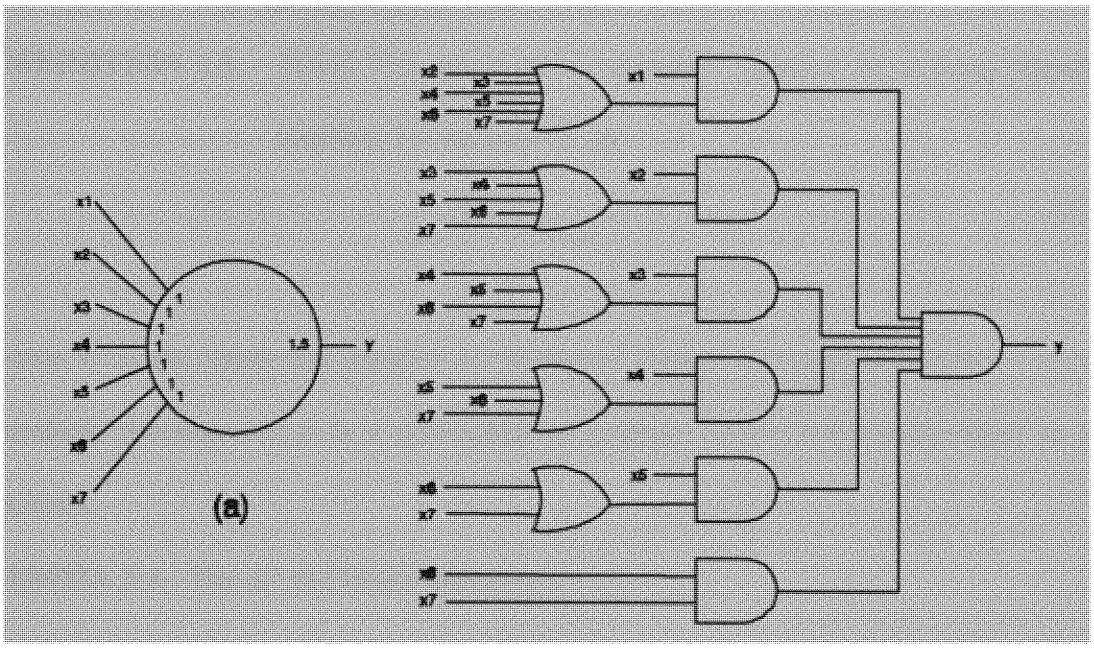


Figure 1.3. Realization of the logic function y by using (a) one threshold gate and (b) AND-OR gates

If we realize this function by using AND-OR logic, we require 12 gates and total of 38 inputs. But if we use threshold gate, we only need a single threshold gate and 7 inputs as shown in Figure 1.3.

Obviously, it becomes clearly when we use the threshold gate to represent a Boolean function than the AND-OR gates. And threshold gate is considerable saving in numbers of gates, components, and inputs.

1.3 Fuzzy threshold functions

Fuzzy threshold functions can be defined using the concepts and techniques developed in fuzzy logic and fuzzy languages.

DEFINITION 1.3. [5,13] The grade of membership function of a fuzzy threshold function f is defined to be:

$$\mu_T(f) = 1-\rho C \qquad [1.8]$$

where C is the minimum number of minterms change in order to convert f to be a threshold function and ρ is a normalization constant which will be discussed in later sections.

LEMMA 1.1. For all threshold functions f_T,

$$C(f_T) = 0 \qquad [1.9]$$

$$\mu_T(f_T) = 1 \qquad [1.10]$$

$\mu_T(f_T)=1$ indicates that f_T has full membership in the class of threshold functions.

LEMMA 1.2. For all functions f of n variables, an upper bound for C is:

$$C(n) \leq \frac{No.\ of\ total\ minterms}{2} \qquad [1.11]$$

$$C(n) \leq 2^{n-1} \qquad [1.12]$$

LEMMA 1.3. For all functions f of n variables, an upper bound for C can be improved to be:

$$C(n) \leq \min(\#ofminterms, \#oftotalminterms - \#ofminterms) \quad [1.13]$$

Proof. If the number of minterms is less than half of the total minterms, then we can change all the minterms to zeros. Thus, we will be able to obtain a threshold function. If the number of minterms is more than half of the total minterms, then we can change all the maxterms to ones. Thus, we will be able to obtain a threshold function.

LEMMA 1.4. A function consists of only a single minterm or a function of single maxterm is always a threshold function.

LEMMA 1.5. The function f is a threshold function if and only if C(n)=0.

LEMMA 1.6. Thus, a better upper bound for C may be obtained as:

$$C(n) \leq \min(\#.ofminterms, \#.oftotalminterms - \#.ofminterms) - 1 \quad [1.14]$$

Proof. We can cut the upper bound by one by changing all the logic zeros to logic ones except one minterm if there are more minterms or vice versa.

1.4 Determination of ρ_n

1.4.1 For two variables (n=2)

The Boolean lattice representation of two variables is shown in Figure 1.4. These four vertices represent the minterms $x_1'x_0'$, $x_1'x_0$, x_1x_0' and x_1x_0 which are represented as

(0,0), (0,1), (1,0), and (1,1) respectively. The minterm numbers 0, 1, 2, and 3 are also indicated.

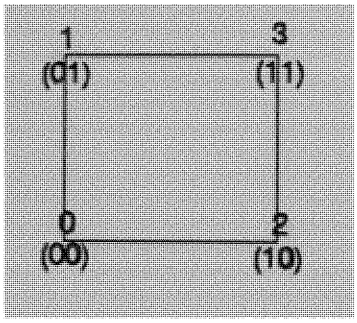


Figure 1.4. A Boolean lattice representation of two variables

Example 1.8. The Exclusive OR function

$$f_{XOR} = \Sigma(1,2)$$

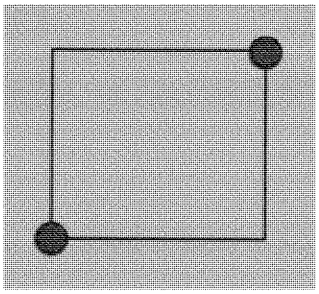


Figure 1.5. A Boolean lattice representation of two variables of $f_{XOR} = \Sigma(1,2)$

This is not a threshold function and the minimum number of minterms to be changed is equal to one if we want to change f_{XOR} to a threshold function. So $C(f_{XOR})=1$.

Example 1.9. The Exclusive NOR function

$$f_{EQU} = \Sigma(0,3)$$

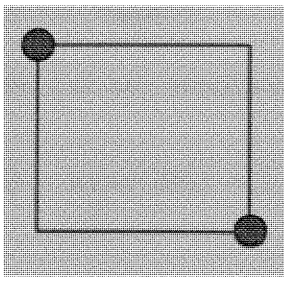


Figure 1.6. A Boolean lattice representation of two variables of $f_{EQU} = \Sigma(0,3)$

f_{EQU} is not a threshold function and it is a complement of f_{XOR}. The minimum number of changes of minterms is equal to one if we want to change f_{EQU} to be a threshold function. So $C(f_{EQU}) = 1$.

From the above two examples, we can conclude that all functions of two variables have the following properties.

DEFINITION 1.4. A function is called a most inseparable function f_{MIS} if and only if a function f of n variables satisfied the following condition.

$$C(f_{MIS}) = \max_{\forall f}\{C(n)\} \qquad [1.15]$$

in order to normalized μ_T to be in the interval [0 1]. And we define the grade of membership of f_{MIS} to be zero.

$$\mu_T(f_{MIS}) = 0 \qquad [1.16]$$

Corollary 1.1. By using definition 1.4.,

$$\rho_n = \frac{1}{C_{max}(n)} \qquad [1.17]$$

THEOREM 1.4. For all functions f of two variables, the max C(n) is equal to 1.

$$C_{max}(2) = \max_{f}\{C(n)\} = 1 \qquad [1.18]$$

and also denoted as $C_{max}(2) = 1$

THEOREM 1.5. For all functions f of two variables, the normalization constant ρ_2 can be obtained by as follows.

$$\mu_T(f_{MIS}) = 1-\rho_2 C_{max} \qquad [1.19]$$

$$0 = 1-\rho_2 C_{max} \qquad [1.20]$$

$$\rho_2 = \frac{1}{C_{max}(2)} = 1 \qquad [1.21]$$

THEOREM 1.6. For all functions f of two variables, the grade of membership function is:

$$\mu_T(f) = 1-\frac{C}{C_{max}(2)} = 1-C \qquad [1.22]$$

1.4.2 For three variables (n=3)

The Boolean lattice representation of three variables is shown in Figure 1.7. These eight vertices represent the minterms $x_2'x_1'x_0'$, $x_2'x_1'x_0$, $x_2'x_1x_0'$, $x_2'x_1x_0$, $x_2x_1'x_0'$, $x_2x_1'x_0$, $x_2x_1x_0'$ and $x_2x_1x_0$ which are represented as (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0) and (1,1,1) respectively. The minterm numbers 0, 1, 2, 3, 4, 5, 6, and 7 are also indicated.

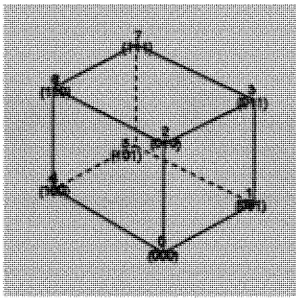


Figure 1.7. A Boolean lattice representation of three variables

Example 1.10. $f_1 = \Sigma(0,3,4,6)$

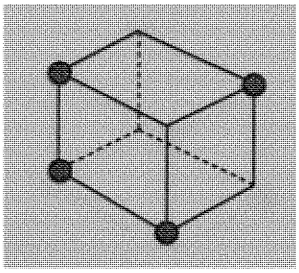


Figure 1.8. A Boolean lattice representation of three variables of $f_1 = \Sigma(0,3,4,6)$

This is not a threshold function and the minimum number of changing total minterms is equal to one if we want to change it to be a threshold function. So $C(f_1)=1$.

Note. Odd function and Even function

Odd function: The minterms of a function are unit distance apart from each other and each binary value has an odd number of 1's.

Even function: The complement of an odd function is an even function and each binary value of these minterms has an even number of 1's.

Example 1.11. A map for a three-variable Exclusive OR function is shown in Figure 1.9.

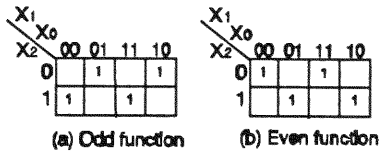


Figure 1.9. A map for a three-variable Exclusive OR function

Example 1.12. $f_{odd}(3) = \Sigma(1,2,4,7)$

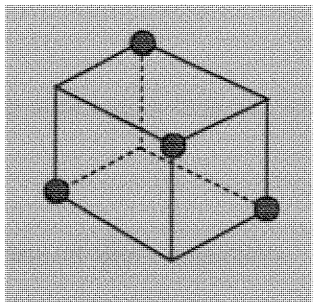


Figure 1.10. A Boolean lattice representation of three variables of $f_{odd}(3) = \Sigma(1,2,4,7)$

This is not a threshold function and the minimum number of changing total minterms is equal to two if we want to change it to be a threshold function. So $C(f_{odd}(3))=2$.

Example 1.13. $f_{even}(3) = \Sigma(0,3,5,6)$

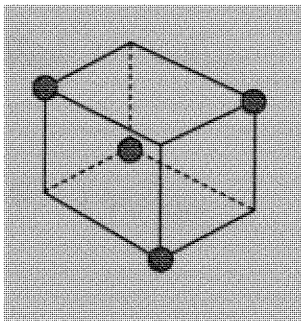


Figure 1.11. A Boolean lattice representation of three variables of $f_{even}(3) = \Sigma(0,3,5,6)$

This is also not a threshold function and it is a complement function of $f_{odd}(3)$. The minimum number of changing total minterms is equal to two if we want to change it to be a threshold function. So $C(f_{even}(3)) = 2$.

From the above, it can be concluded that all functions of two variables have the following properties.

THEOREM 1.7. For all functions of three variables,

$$\max_{\forall f}\{C(n)\} = C_{max} \qquad [1.23]$$

THEOREM 1.8. For all functions f of three variables, the normalization constant ρ_3 can be obtained by as follows.

$$C_{max}(3) = 2 \quad [1.24]$$

$$\mu_T(f_{MIS}) = 1-\rho_3 C_{max} \quad [1.25]$$

$$0 = 1-\rho_3 C_{max} \quad [1.26]$$

$$\rho_3 = \frac{1}{C_{max}(3)} = \frac{1}{2} \quad [1.27]$$

THEOREM 1.9. For all functions of three variables, the grade of membership function is:

$$\mu_T(f) = 1-\frac{C}{C_{max}(3)} = 1-\frac{C}{2} \quad [1.28]$$

1.4.3 For n variables

THEOREM 1.10. From the derivation of two and three variables, we can generalize the grade of membership function for n variables as:

$$\mu_T(f(x_1,x_2,\ldots,x_n)) = 1-\frac{C}{C_{max}(n)} \quad [1.29]$$

1.4.4 Summary

From above theorems and examples, we can summarize the results in Table 1.2 and derive an algorithm for determining $C_{max}(n)$, an algorithm for determining C(n), and an algorithm for generating most inseparable functions of subsequent sections.

No. of variables	C_{max}	No. of most inseparable functions	most inseparable functions	membership function	possible values of μ_T
2	1	2	$\Sigma(0,3)$ $\Sigma(1,2)$	$\mu_T(f)=1-C$	0,1
3	2	2	$\Sigma(0,3,5,6)$ $\Sigma(1,2,4,7)$	$\mu_T(f)=1-C/2$	0,1/2,1

Table 1.2. Summary of two and three variables functions

1.5 An algorithm for determining $C_{max}(n)$

After the derivations of two and three variables, we can determine $C_{max}(n)$ for n variables by using the same method.

THEOREM 1.11. For all functions of n variables with n≥4,

$$C_{max}(n) = 2^{n-1}-1 \qquad [1.30]$$

THEOREM 1.12. For all member of variables, the number of most inseparable functions are always equal to 2.

THEOREM 1.13. The function f is a threshold function if and only if f' is a threshold function. This theorem can be generalized to fuzzy threshold functions.

THEOREM 1.14. For all functions f of n variables, then

$$C(f) = C(f') \qquad [1.31]$$

$$\mu_T(f) = \mu_T(f') \qquad [1.32]$$

1.6 An algorithm for determining C(n)

Boolean lattice and Karnaugh map are both common ways to represent logic functions. So we replace Boolean lattice by using Karnaugh map to do the algorithm for determining C(n) in this section.

We limit our presentation in two and three variables here and it is the same approach while above three variables.

We can convert one representation to the other by using Figures 1.12 and 1.13.

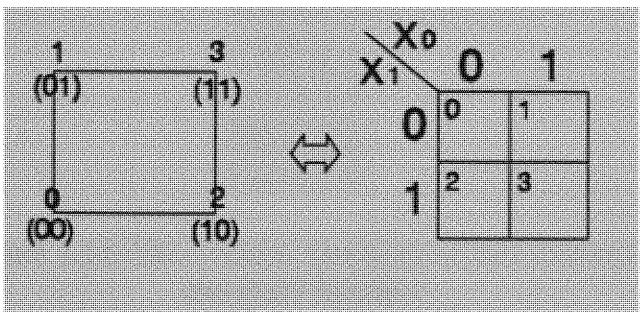


Figure 1.12. The two-variable map from Boolean lattice representation to Karnaugh map

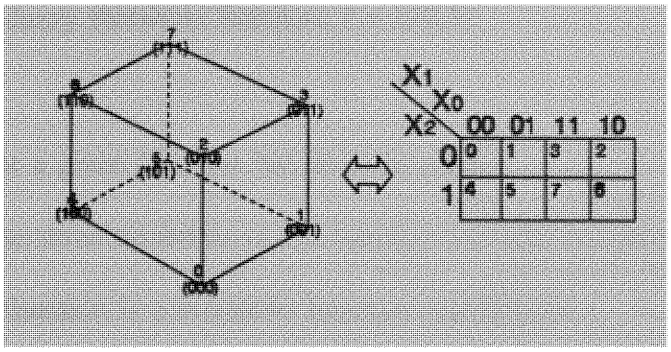


Figure 1.13. The three-variable map from Boolean lattice representation to Karnaugh map

LEMMA 1.7. If f_{min} is a minterm function, then $C(f_{min}) = 0$.

If f_{max} is a minterm function, then $C(f_{max}) = 0$.

LEMMA 1.8. If a given function has two minterms and the two minterms are adjacent on the Karnaugh map, then the function is a threshold function and C(n)=0. Otherwise the function is not a threshold function and C(n)=1.

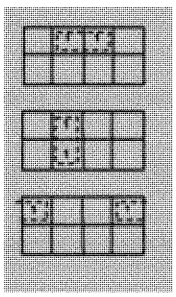


Figure 1.14. The possible positions of two minterms threshold function

Example 1.14. A threshold function: f = Σ(0,1)

A fuzzy threshold function: f = Σ(2,5)

LEMMA 1.9. For a given function has three minterms,

Case 1. If all these three minterms of the function are adjacent on the Karnaugh map, then the function is a threshold function and C(n)=0.

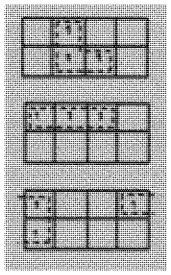


Figure 1.15. The possible positions of three minterms threshold function

Example 1.15. A threshold function: f = Σ(0,4,5)

Case 2. If only two minterms of the function are adjacent on the Karnaugh map, then the function is not a threshold function and C(n)=1.

Example 1.16. A fuzzy threshold function: f = Σ(1,3,6)

Case 3. If all these three minterms are not adjacent on the Karnaugh map, then the function is not a threshold function and C(n)=1.

Example 1.17. A fuzzy threshold function: f = Σ(1,2,7)

LEMMA 1.10. For a given function has four minterms,

Case 1. If all these four minterms of the function are adjacent on the Karnaugh map, then the function is a threshold function and C(n)=0.

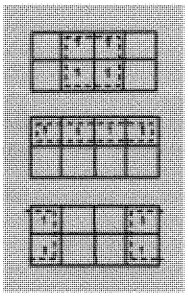


Figure 1.16. The possible positions of four minterms threshold function

Example 1.18. A threshold function: f = Σ(0,1,4,5)

Case 2. If three minterms of the function are adjacent on the Karnaugh map, then the function is not a threshold function and C(n)=1.

Example 1.19. A fuzzy threshold function: f = Σ(0,1,4,7)

Case 3. If there are two groups and two minterms of each group are adjacent, then the function is not a threshold function and C(n)=1.

Example 1.20. A fuzzy threshold function: f = Σ(0,1,6,7)

Case 4. If all these four minterms are not adjacent on the Karnaugh map, then the function is not a threshold function and C(n)=2 (in three variables case).

Example 1.21. A fuzzy threshold function: f = Σ(0,3,5,6)

According to above Lemmas and examples, we can conclude an algorithm for determining C(n) as follows.

We can obtain C(n) by finding the minimum of these three cases as indicated in Figure 1.17.

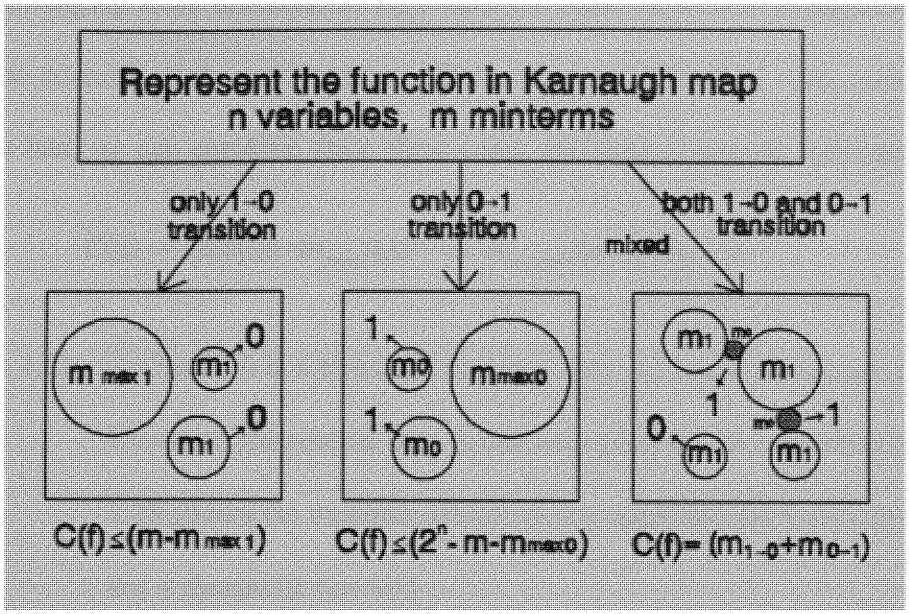


Figure 1.17. The algorithm for determining C(n)

Case 1. Changing only logic one to logic zero transition.

Step 1. To find the largest threshold subfunction f_{max1}. Let the number of minterms in f_{max1} equal to m_{max1}.

Step 2. To change the rest of minterms to logic zero.

Step 3. Thus, we can obtain $C(n) \leq m\text{-}m_{max1}$.

Case 2. Changing only logic zero to logic one transition

Step 1. To find the largest threshold subfunction f_{max0}. Let the number of minterms in f_{max0} equal to m_{max0}.

Step 2. To change the rest of minterms to logic one.

Step 3. Thus, we can obtain $C(n) \leq 2^n\text{-}m\text{-}m_{max0}$.

Case 3. Changing some minterms to logic zero and some maxterms to logic one in order to connect all the subfunctions. Thus, we can obtain $C(n) = m_{1\to0}+m_{0\to1}$.

At last, we can combine these three cases and then obtain the minimum C(n).

$$C(n) = \min[(m-m_{max1}),(2^n-m-m_{max0}),(m_{1\to0}+m_{0\to1})] \quad [1.33]$$

LEMMA 1.11. If a function is a threshold function, then all the minterms are connected. However, the reverse may not be true as shown in example 1.22. Figure 1.18 shows the example which all the minterms are connected, but it is not a threshold function.

Example 1.22.

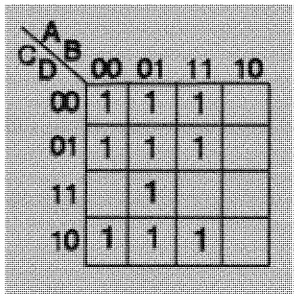


Figure 1.18. A fuzzy threshold function

1.7 An algorithm for generating most inseparable functions f_{MIS}

When we have most inseparable functions of two variables, we can generate most inseparable functions of three, four, five variables, etc. Figure 1.19 shows the algorithm for generating most inseparable functions f_{MIS}.

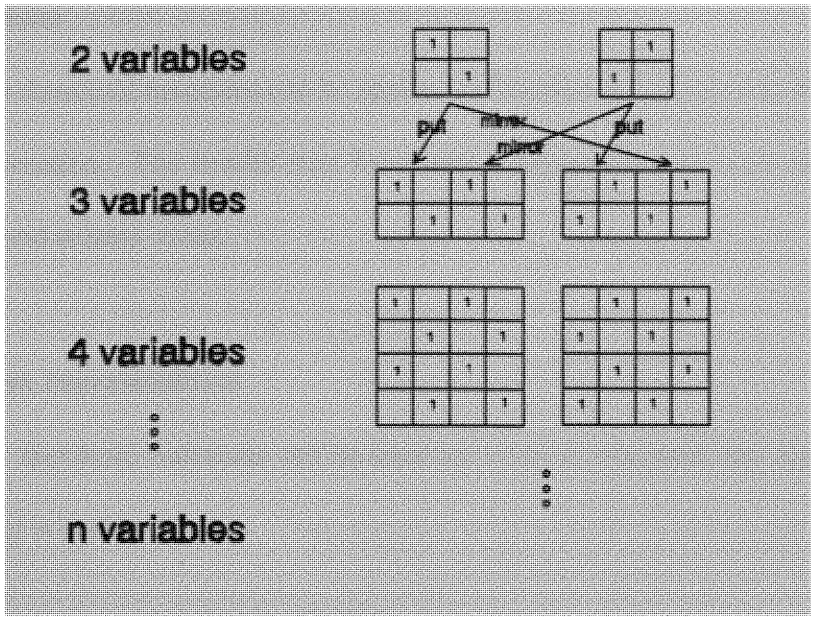


Figure 1.19. The algorithm for generating most inseparable functions f_{MIS}

THEOREM 1.15. For n variables, there are only two most inseparable functions (f_{MIS}) and denoted as $f_{even}(n)$ and $f_{odd}(n)$. Their relationship is:

$$f_{even}(n) = f'_{odd}(n) \qquad [1.34]$$

THEOREM 1.16. Given $f_{even}(n)$ and $f_{odd}(n)$, $f_{even}(n+1)$ and $f_{odd}(n+1)$ can be generated as follows.

$$f_{even}(n+1) = x'_{n+1} f_{even}(n) + x_{n+1} f_{odd}(n) \qquad [1.35]$$

$$f_{odd}(n+1) = x_{n+1} f_{even}(n) + x'_{n+1} f_{odd}(n) \qquad [1.36]$$

1.8 Applications

Fuzzy threshold functions can be applied to function representation, data reduction, and error correction. The results may have useful applications to logic design, pattern recognition, and related areas.

1.8.1 Function representation

Fuzzy threshold functions can be used as a tool to represent any logic function f.

If C(n) = i, then

$$f = f_T \oplus \Sigma(m_1,m_2,...,m_i) \qquad [1.37]$$

$$f_T = f \oplus \Sigma(m_1,m_2,...,m_i) \qquad [1.38]$$

where f_T is a threshold function, $\oplus$ indicates the exclusive-or operation and m_1, m_2,..., m_i are minterms which should be changed from f_T in order to obtain f.

Usually, this representation is not unique. In other word, there may exist f'_T such that:

$$f = f'_T \oplus \Sigma(m'_1,m'_2,...,m'_i) \qquad [1.39]$$

Example 1.23. $f = \Sigma(0,3,5,6)$ is an inseparable function with C(f)=2, it can be represented as $f = \Sigma(0,2,3,6)\oplus\Sigma(m_2,m_5)$ or $f = \Sigma(0,1,3,5)\oplus\Sigma(m_1,m_6)$ where $\Sigma(0,2,3,6)$ and $\Sigma(0,1,3,5)$ both are threshold functions.

1.8.2 Data compression

A threshold function can be represented $[w_1,w_2,...,w_n;T]$ which has a simpler representation compare with a general logic function.

For function f with C(n)=1, then function f can be represented by:

$$f = f_T \oplus \{m_i\} \qquad [1.40]$$

Thus, for transmitting f, instead of transmitting the total truth table or the corresponding Karnaugh map, we can transmit only f_T and the minterms m_i. Therefore, data compression may be achieved.

1.8.3 Error detection and correction

Fuzzy threshold functions also can be applied to error detection. For example, if we know that a most inseparable function (f_{MIS}) is transmitted and a single error occurred, then at the receiving end,

$$C(f'_{MIS}) \neq C_{max} \qquad [1.41]$$

where f'_{MIS} indicates the function received.

We know that there are only two most inseparable functions for n variables. Therefore, if we transmit a most inseparable function of n variables, then all the single error can be detected and corrected. Actually, the capability of error correction is stronger than when single error occurred.

$$f_{MIS}(n) = \Sigma(w_0,w_1,w_2,\ldots,w_n;C_{max}(f_{MIS})) \quad [1.42]$$

Example 1.24. $f_{MIS} = \Sigma(0,3,5,6;2)$ is transmitted, $f = \Sigma(0,2,3,6;0)$ is received. $C_{MIS}(f_{MIS})=2$ is not equal to C(n)=0. So the numbers of minterms 2 and 5 of f are error and they can be corrected.

THEOREM 17. The Hamming distance between $\{f_{even}(n), f_{odd}(n)\}$ of two most inseparable functions is equal to 2^n. So using minimum distance coding, 2^{n-1}-1 errors can be corrected if the words are $\{f_{even}(n), f_{odd}(n)\}$.

THEOREM 18. Since we only correct a single error of a most inseparable function, the number of minterms which are not changed is equal to 2^{n-1}-1.

1.9 Conclusion

The technique of applying fuzzy logic to threshold functions are introduced. There are many new results and examples are given to illustrate them. The algorithm of determining the minimum number of minterms changes C(n) in order to convert f into a threshold function f_T, and finding C_{max} are investigated. We also found there are two most inseparable functions and they are odd function and even function. Also, the applications of function representation, data reduction, and error correction are presented. The topics considered in this chapter appear apparently to be a fertile field for further study. The results also may have useful applications in logic design, pattern recognition, and related areas.

Chapter Two

Fuzzy Monotone Functions and Applications

2.1 Introduction

Monotone functions play an important role in switching theory, logic design, computer design, flash analog to digital converter, pattern classification, and pattern recognition. In particular, flash analog to digital converter is designed using the set of all monotone increasing (decreasing) functions. The details shall be discussed in chapter 4 and 5.

In this chapter, I will introduce fuzzy monotone functions which are using the concepts and techniques developed in fuzzy logic and fuzzy languages.

2.2 Monotone increasing functions f_{MINC}

DEFINITION 2.1. [9,15] A Boolean function f of n variables is *monotone increasing* if and only if $x \leq y$ implies that $f(x) \leq f(y)$, where $x=(x_1,x_2,...,x_n)$ and $y=(y_1,y_2,...,y_n)$ and $x_1, x_2,..., x_n, y_1, y_2,..., y_n$ are either "logic zero" or "logic one".

THEOREM 2.1. [9,15] For a function which is n variables, there are 2^n+1 monotone increasing functions. More specifically, they are shown in Table 2.1.

X_n X_{n-1} ... X_2 X_1	f_1 f_2 f_3 ... f_{2^n} f_{2^n+1}
0 0 ... 0 0	0 0 0 ... 0 1
0 0 ... 0 1	0 0 0 ... 1 1
0 0 ... 1 0	0 0 0 ... 1 1
0 0 ... 1 1	0 0 0 ... 1 1
⋮ ⋮ ⋮ ⋮ ⋮	⋮ ⋮ ⋮ ⋮ ⋮
1 1 ... 0 1	0 0 0 ... 1 1
1 1 ... 1 0	0 0 1 ... 1 1
1 1 ... 1 1	0 1 1 ... 1 1

Table 2.1. All monotone increasing functions of n variables

DEFINITION 2.2. From Table 2.1, a set of all monotone increasing functions can be defined as f_{MINC}.

$$f_{MINC} = \{f_1, f_2, f_3, \ldots, f_{2^n+1}\} \qquad [2.1]$$

2.3 Fuzzy monotone increasing functions

After the discussion of monotone increasing functions, fuzzy monotone increasing functions can be defined by the concepts of fuzzy logic and fuzzy languages.

DEFINITION 2.3. [5,13] The grade of membership of a fuzzy monotone increasing function $f(x_1,x_2,...,x_n)$ is defined to be:

$$\mu_{MINC}(f(x_1, x_2, \ldots x_n)) = 1-\rho_n C \qquad [2.2]$$

where C is the minimum number of minterms change in order to convert $f(x_1,x_2,...,x_n)$ to be a monotone increasing function (a member in f_{MINC}) and ρ_n is a normalization constant which will be determined in section 2.4.

LEMMA 2.1. For a monotone increasing function $f(x_1,x_2,...,x_n)$, the minimum number of minterms change is equal to:

$$C(f(x_1, x_2, \ldots, x_n)) = 0 \qquad [2.3]$$

LEMMA 2.2. For a monotone increasing function $f(x_1,x_2,...,x_n)$, $f(x_1,x_2,...,x_n)$ has full membership in the class of monotone functions and indicates as:

$$\mu_{MINC}(f(x_1, x_2, \ldots, x_n)) = 1 \qquad [2.4]$$

And we normalize μ_{MINC} in the interval [0 1].

2.4 Determination of ρ_n

According to Table 1, for all monotone increasing functions, if we scan the output column from the top to the bottom, there is only one change of truth value from 0 to 1.

DEFINITION 2.4. Thus, we define that the function is most dissimilar to monotone increasing functions to be the most variation function f_{MVF} which has the following property, that is, if we scan the output column from the top to the bottom, there will be most number of changes of truth value from 0 to 1.

$$C_{MVF}(f(x_1, x_2, \ldots, x_n)) = \max_{\forall f}\{C_{MINC}(f(x_1, x_2, \ldots, x_n)\} \quad [2.5]$$

From definition 2.4, it implies that the two most variation functions are as Table 2.2.

f_{MVF1}	f_{MVF2}
0	1
1	0
0	1
1	0
.	.
.	.
.	.
0	1
1	0

Table 2.2. The two most variation functions f_{MVF1} and f_{MVF2}

2.4.1 For two variables (n=2)

THEOREM 2.2. For all functions f of two variables, the maxium value $C(f(x_1,x_2))$ is equal to 2.

$$C_{max}(2) = \max_{f}\{C_{MINC}(f(x_1, x_2))\} = 2 \quad [2.6]$$

THEOREM 2.3. For all functions f of two variables, the normalization constant ρ_2 can be obtained by as follows.

$$\mu_{MVF}(f(x_1, x_2)) = 1 - \rho_2 C_{max}(2) \quad [2.7]$$

$$0 = 1-\rho_2 C_{\max}(2) \qquad [2.8]$$

$$\rho_2 = \frac{1}{C_{\max}(2)} = \frac{1}{2} \qquad [2.9]$$

THEOREM 2.4. For all functions f of two variables, the grade of membership function is:

$$\mu_{MINC}(f(x_1, x_2)) = 1-\frac{C}{C_{\max}(2)} = 1-\frac{C}{2} \qquad [2.10]$$

2.4.2 For three variables (n=3)

THEOREM 2.5. For all functions f of three variables, the normalization constant ρ_3 can be obtained by as follows.

$$C_{\max}(3) = 4 \qquad [2.11]$$

$$\mu_{MVF}(f(x_1, x_2, x_3)) = 1-\rho_3 C_{\max}(3) \qquad [2.12]$$

$$0 = 1-\rho_3 C_{\max}(3) \qquad [2.13]$$

$$\rho_3 = \frac{1}{C_{\max}(3)} = \frac{1}{4} \qquad [2.14]$$

THEOREM 2.6. For all function of three variables, the grade of membership function is:

$$\mu_{MINC}(f(x_1, x_2, x_3)) = 1 - \frac{C}{C_{\max}(3)} = 1 - \frac{C}{4} \quad [2.15]$$

2.4.3 For n variables

THEOREM 2.7. From above derivation, for all functions f of n variables can be generalized as:

$$C_{\max}(n) = 2^{n-1} \quad [2.16]$$

$$\rho_n = \frac{1}{C_{\max}(n)} \quad [2.17]$$

$$\mu_{MINC}(f(x_1, x_2, \ldots, x_n)) = 1 - \frac{C}{C_{\max}(n)} \quad [2.18]$$

2.4.4 Summary

From above theorems and examples, the results can be summarized as following theorems, and tables. Furthermore, the algorithm for determining $C_{max}(n)$ and C(n), and the algorithm for generating most variation functions are also can be achieved by using these results.

For any two and three-variable functions, the results can be summarized as Table 2.3.

No. of variables	C_{max}	No. of most variation functions	most variation functions	membership function	possible values of μ_{MINC}
2	2	2	$\Sigma(0,2)$ $\Sigma(1,3)$	$\mu_{MINC}(f)$ = 1-(C/2)	0,1/2,1
3	4	2	$\Sigma(0,2,4,6)$ $\Sigma(1,3,5,7)$	$\mu_{MINC}(f)$ = 1-(C/4)	0,1/4,2/4 ,3/4,1

Table 2.3. Summary of two and three-variable functions

THEOREM 2.8. The number of monotone increasing function (f_{MINC}) is 2^n+1.

And it can be summarized as shown in Table 2.4.

number of variables	number of f_{MINC}
2	5
3	9
4	17
5	33
. . .	. . .
n	2^n+1

Table 2.4. Summary of f_{MINC} of n variables

LEMMA 2.3. A function of constant "logic zeros" or constant "logic ones" is a monotone increasing function.

Example 2.1. For a function of three variables, then $f=\Sigma(0,1,2,3,4,5,6,7)$ and $f=\prod(0,1,2,3,4,5,6,7)$ are two monotone increasing functions.

THEOREM 2.9. Given two functions f1 and f2 which both are monotone increasing functions of n variables, then $f1_{MINC} \cdot f2_{MINC}$ is a monotone increasing function which is equal to the smaller one.

Example 2.2. Two functions $f1_{MINC}=\Sigma(5,6,7)$ and $f2_{MINC}=\Sigma(4,5,6,7)$ which both are monotone increasing functions of three variables , then $f1_{MINC} \cdot f2_{MINC}$ is a monotone increasing function which is equal to the smaller one $f1_{MINC}=\Sigma(5,6,7)$.

THEOREM 2.10. Given two functions f1 and f2 which both are monotone increasing functions of n variables, then $f1_{MINC}+f2_{MINC}$ is a monotone increasing function which is equal to the larger one.

Example 2.3. Two functions $f1_{MINC}=\Sigma(5,6,7)$ and $f2_{MINC}=\Sigma(4,5,6,7)$ which both are monotone increasing functions of three variables , then $f1_{MINC}+f2_{MINC}$ is a monotone increasing function which is equal to the larger one $f2_{MINC}=\Sigma(4,5,6,7)$.

2.5 An algorithm for determining $C_{max}(n)$

THEOREM 2.11. For all functions of two variables, $C_{max}(2) = 2$

THEOREM 2.12. For all functions of three variables, $C_{max}(3) = 4$

THEOREM 2.13. For all functions of n variables,

$$C_{max}(n) = 2^{n-1} \qquad [2.19]$$

And it can be summarized as shown in Table 2.5.

number of variables	C_{max}	$\rho_n=1/C_{max}$	$\mu_{MINC}(f)=(1-C/C_{max}(n))$
2	2	1/2	1-(C/2)
3	4	1/4	1-(C/4)
4	8	1/8	1-(C/8)
5	16	1/16	1-(C/16)
. . .	. . .	. . .	. . .
n	2^{n-1}	$1/2^{n-1}$	$1-(C/2^{n-1})$

Table 2.5. Summary of n variables

2.6 An algorithm for determining C(n)

For a given function with n variables, we use two pointers, high pointer (HPT) and low pointer (LPT), with starting positions at the first bit and the last bit of the output of the truth table respectively. The basic idea of the algorithm is that HPT is moving Downward and The LPT is moving upward. In addition, we move HPT and LPT alternatively until they meet.

The Flow Chart is shown in Figure 2.1.

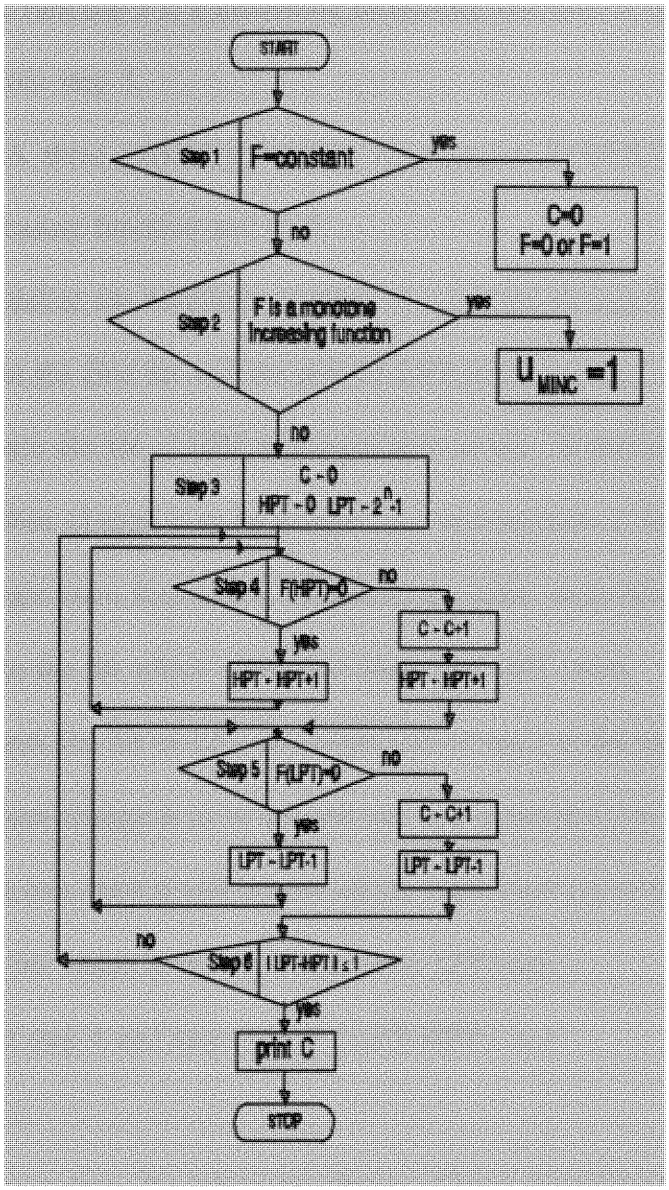


Figure 2.1. Flow Chart for determining C(n)

Step 1. Check all the minterms of a function are logic zeros or logic ones. If yes, the function is a monotone increasing function..

Example 2.4. A three variables function $f(x_1,x_2,x_3) = \Sigma(0,1,2,3,4,5,6,7)$. It means all the minterms are logic ones. So C=0 and f is a monotone increasing function.

Step 2. Check function f is a monotone increasing function or not. If yes, μ_{MINC}=1. If not, goto step 3.

Example 2.5. A three variables function $f(x_1,x_2,x_3) = \Sigma(4,5,6,7)$. This function is a monotone increasing function and C=0.

Step 3. To initialize the minimum number of minterms change C to 0, the two pointers HPT at the first bit and the pointer LPT at the last bit.

Step 4. To check the bit of HPT is a logic zero or not. If the bit of HPT is not a logic zero, then change it to be a logic zero and increase C by 1. If the bit of HPT is a logic zero, then it will bypass all the logic zeros which below HPT and set HPT to the bit which is the first logic one HPT meet.

Step 5. To check the bit of LPT is a logic one or not. If the bit of LPT is not a logic one, then change it to be a logic one and increase C by 1. If the bit of LPT is a logic one, then it will bypass all the logic ones which above LPT and set LPT to the bit which is the first logic zero LPT meet.

Step 6. Repeat step 4 and step 5 until $|\text{LPT-HPT}| \leq 1$.

After these steps, the minimum changes C(n) can be found and it can be changed to be a monotone increasing function.

Example 2.6. Given a three variables function $f=\Sigma(1,4,5,7)$.

(1) To initialize C to 0, HPT to the first bit 0, and LPT to the last bit 7.

(2) To check the bit 0 of HPT is a logic zero or not. In this example, it is a logic zero. So HPT will go to the bit 1, the first logic one HPT meet.

(3) To check the bit 7 of LPT is a logic one or not. In this example, it is a logic one. So LPT will go to the bit 6, the first logic zero LPT meet.

(4) The bit 1 of HPT is a logic one. So change bit 1 to be a logic zero and set HPT to the next bit 2, the first logic zero HPT meet.

(5) The bit 6 of LPT is a logic zero. Then it will be changed to be a logic one and set LPT to the bit 5, the first logic one LPT meet.

(6) The bit 2 of HPT is a logic zero, and so is bit 3. Then HPT will bypass bit 3 and go to bit 4.

(7) The bit 5 of LPT is a logic one, and so is bit 4. Then LPT will bypass bit 4 and go to bit 3.

(8) Now pointer HPT is located at bit 4 and pointer LPT is located at bit 3. $|LPT-HPT| \leq 1$, So the algorithm will stop and C(n) is equal to 2.

As a result, we found the minimum changes C(n) is equal to 2 in order to change f to become as a monotone increasing function. And function f become as $f=\Sigma(4,5,6,7)$.

2.7 An algorithm for generating most variation functions f_{MVF1} and f_{MVF2}

When we have most variation functions of two variables, we can generate most variation functions of three, four, five variables, etc. Figure 2.2. shows the algorithm for generating most variation functions f_{MVF}.

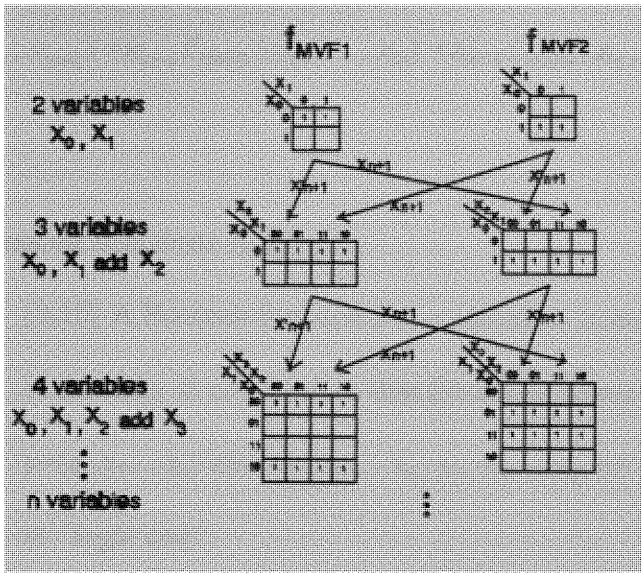


Figure 2.2. An algorithm for generating most variation functions f_{MVF}

THEOREM 2.14. For n variables, there are only two most variation functions (f_{MVF}) and denoted as $f_{MVF1}(n)$ and $f_{MVF2}(n)$. And their relationship is:

$$f_{MVF1}(n) = f'_{MVF2}(n) \qquad [2.20]$$

Example 2.7. For a three-variable function, $f_{MVF1}=\Sigma(0,2,4,6)$ and $f_{MVF2}=\Sigma(1,3,5,7)$. $f_{MVF1}(3)=f_{MVF2}'(3)$.

THEOREM 2.15. Let the additional variables be x_{n+1}, then $f_{MVF1}(x_1,x_2,...,x_n,x_{n+1})$ can be generated recursively as follows.

$$f_{MVF1}(x_1,\ldots,x_n,x_{n+1}) = x'_{n+1} f_{MVF1}(x_1,\ldots,x_n) + x_{n+1} f_{MVF2}(x_1,\ldots,x_n) \qquad [2.21]$$

THEOREM 2.16. Let the additional variables be x_{n+1}, then $f_{MVF2}(x_1,x_2,...,x_n,x_{n+1})$ can be generated recursively as follows.

$$f_{MVF2}(x_1, \ldots, x_n, x_{n+1}) = x_{n+1} f_{MVF1}(x_1, \ldots, x_n) + x'_{n+1} f_{MVF2}(x_1, \ldots, x_n) \quad [2.22]$$

2.8 Monotone decreasing functions f_{MDEC}

DEFINITION 2.5. [9,15] A Boolean function f of n variables is *monotone decreasing* if and only if $x \geq y$ implies that $f(x) \geq f(y)$, where $x=(x_1,x_2,\ldots,x_n)$ and $y=(y_1,y_2,\ldots,y_n)$ and $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$ are either "logic zero" or "logic one".

X_n	X_{n-1}	$\cdots$	X_2	X_1	f_1	f_2	f_3	$\cdots$	f_{2^n}	f_{2^n+1}
0	0	$\cdots$	0	0	0	1	1	$\cdots$	1	1
0	0	$\cdots$	0	1	0	0	1	$\cdots$	1	1
0	0	$\cdots$	1	0	0	0	0	$\cdots$	1	1
0	0	$\cdots$	1	1	0	0	0	$\cdots$	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	$\cdots$	0	1	0	0	0	$\cdots$	1	1
1	1	$\cdots$	1	0	0	0	0	$\cdots$	1	1
1	1	$\cdots$	1	1	0	0	0	$\cdots$	0	1

Table 2.6. All monotone decreasing functions of n variables

2.8.1 Fuzzy monotone decreasing functions

The concepts and techniques of monotone decreasing functions are also using fuzzy logic, fuzzy languages, and fuzzy neural networks. All the results are similar to monotone increasing functions. As a result, I do not discuss the details in this section

again and only write down the important results here.

Fuzzy monotone decreasing functions is also defined using the concept of fuzzy logic and fuzzy languages.

$$\mu_{MDEC}(f(x_1, x_2, \ldots, x_n)) = 1-\rho_n C \qquad [2.23]$$

where we normalize μ_{MDEC} in the interval [0 1] and $\mu_{MDEC} = 1$ for all the monotone decreasing functions.

THEOREM 2.17. For all function of n variables, the number of monotone decreasing functions (f_{MDEC}) is equal to 2^n+1.

THEOREM 2.18. For all function f of n variables, the number of maximum number of minterms changes (C_{max}) in order to become as a monotone decreasing functions (f_{MDEC}) is equal to 2^{n-1}.

So the membership function becomes as:

$$\mu_{MDEC} = 1-\frac{C}{C_{max}(n)} = 1-\frac{C}{2^{n-1}} \qquad [2.24]$$

THEOREM 2.19. The two most variation functions f_{MVF1} and f_{MVF2} are the same as those of monotone increasing functions.

The algorithm for determining C(n) is similar to the one of monotone increasing functions except HPT is checking the bit is logic one or not and LPT is checking logic zero or not.

2.9 Applications

Fuzzy monotone increasing functions can be applied to switching theory, logic design, computer design, flash analog to digital converter, pattern classification, pattern recognition, and pattern understanding. In particular, flash analog to digital converter is designed using the set of all monotone increasing (decreasing) functions.

2.9.1 Function representation

Fuzzy monotone increasing functions can be used as a tool to represent any logic function f.

If C(n) = i, then

$$f = f_{MINC} \oplus \Sigma(m_1, m_2, \ldots, m_i) \qquad [2.25]$$

$$f_{MINC} = f \oplus \Sigma(m_1, m_2, \ldots, m_i) \qquad [2.26]$$

where f_{MINC} is a monotone increasing function, $\oplus$ indicates the exclusive or operation and m_1, m_2,..., m_i are minterms which should be changed from f_{MINC} in order to obtain f.

Example 2.8. If we want to convert $f(x_1,x_2,x_3) = \Sigma(1,3,4,6,7)$ into a monotone increasing function $f_{MINC}(x_1,x_2,x_3)$, then $C(f(x_1,x_2,x_3))=2$ and it can be represented as $f(x_1,x_2,x_3) = \Sigma(3,4,5,6,7)\oplus\Sigma(m_1,m_5)$.

2.9.2 Data compression

A monotone increasing function can be represented $[w_1,w_2,...,w_n;C]$ which has a

simpler representation compare with a general logic function.

For function f with C(n)=i, then function f can be represented by

$$f = f_{MINC} \oplus \{m_i\} \qquad [2.27]$$

Thus, for transmitting f, instead of transmitting the total truth table or the corresponding Karnaugh map, we can transmit only f_{MINC} and the minterms m_i. Therefore, data compression may be achieved.

2.9.3 Error detection and correction

Fuzzy monotone increasing functions also can be applied to error detection. For example, if we know that a most variation function (f_{MVF}) is transmitted and errors occurred, then at the receiving end,

$$C(f'_{MVF}) \neq C_{max} \qquad [2.28]$$

where f'_{MVF} indicates the function received.

We know that there are only two most variation functions for n variables. Therefore, if we transmit a most variation function of n variables, then all the errors can be detected and corrected.

Example 2.9. $f_{MVF}(x_1,x_2,x_3) = \Sigma(1,3,5,7;C=4)$ is transmitted, $f(x_1,x_2,x_3) = \Sigma(4,5,6,7;C=0)$ is received. $C(f_{MVF}(x_1,x_2,x_3))=4$ is not equal to $C(f(x_1,x_2,x_3))=0$. So the numbers of minterms 1, 3, 4, and 6 of f are erroneous and they can be corrected.

2.9.4 Analog to digital converter

Given a n-bit flash converter input an analog signal, we can get 2^n digital outputs after 2^n-1 comparators. Since these 2^n digital outputs are all monotone increasing (decreasing) functions, any function can be represented by the combinations of these monotone increasing (decreasing) functions. These details and results shall be discussed and presented in chapter 4 and 5.

2.10 Conclusion

Fuzzy monotone functions are introduced, investigated by using the concepts and techniques which developing in fuzzy logic and applied to function representation, data compression, error correction, and monotone flash analog to digital converter. The algorithm for determining C(n), $C_{max}(n)$ and generating the most variation functions f_{MVF1} and f_{MVF2} are presented. Examples are given to illustrated this fuzzy special function. The are many results and they may be have useful applications in logic design, pattern recognition, and related areas.

Chapter Three

Fuzzy Unate Functions and Applications

3.1 Introduction

After the introduction of threshold functions and monotone functions, my principal goal in this chapter is the development of methods for another special logic functions, unate functions, which also play an important role in switching theory, logic design, computer design, pattern classification, and pattern recognition.

Also, fuzzy unate functions are introduced using the concepts and techniques developed in fuzzy logic and fuzzy languages and also applied to function representation, data compression, and error detection.

3.2 Unate functions f_U

DEFINITION 3.1. [21] A function $f(x_1,x_2,...,x_n)$ is said to be *positive* in a variable x_i if there exists a disjunctive or conjunctive expression for the function in which x_i appears only in uncomplemented form.

DEFINITION 3.2. [21] Analogously, $f(x_1,x_2,...,x_n)$ is said to be *negative* in a variable x_i if there exists a disjunctive or conjunctive expression for the function in which x_i appears only in complemented form.

DEFINITION 3.3. [21] Derived from definition 3.1 and 3.2, if a Boolean function $f(x_1,x_2,...,x_n)$ is either positive or negative in x_i, it means that there is no variable appears both uncomplemented and complemented

when f is written in a minimum sum of products form, then function $f(x_1,x_2,...,x_n)$ is said to be *unate* in x_i and f is a unate function.

Example 3.1. The following are both unate functions:

$$f_1(x_1,x_2,x_3) = x_1+x_2x_3$$

$$f_2(x_1,x_2,x_3) = x_1'x_2'+x_3'$$

Proof: All the variables of f_1 function or f_2 appear no both uncomplemented and complemented. So both functions f_1 and f_2 are unate functions.

Example 3.2. On the other hand, the following functions are not unate:

$$f_3(x_1,x_2,x_3) = x_1x_2+x_1'x_2'x_3$$

$$f_4(x_1,x_2,x_3) = x_1x_2'+x_2x_3$$

Proof: Variables x_1 and x_2 of function f_3 and variable x_2 of function f_4 appear both uncomplemented and complemented. So both functions f_3 and f_4 are not unate.

3.3 Fuzzy unate functions

Fuzzy unate functions can be defined using the concepts and techniques developed in fuzzy logic and fuzzy languages.

DEFINITION 3.4. [5,13] The grade of membership of a fuzzy unate function $f(x_1,x_2,...,x_n)$ is defined to be:

$$\mu_U(f(x_1,x_2,...x_n)) = 1-\rho_n C \qquad [3.1]$$

where C is the minimum number of minterms change in order to convert $f(x_1,x_2,...,x_n)$ to

be a unate function (a member in f_U) and ρ_n is a normalization constant which will be determined in later sections.

LEMMA 3.1. For a unate function $f(x_1,x_2,...,x_n)$, the minimum number of minterms change is equal to:

$$C(f(x_1, x_2, \ldots, x_n)) = 0 \qquad [3.2]$$

LEMMA 3.2. For a unate function $f(x_1,x_2,...,x_n)$, $f(x_1,x_2,...,x_n)$ has full membership in the class of unate functions and indicates as:

$$\mu_U(f(x_1, x_2, \ldots, x_n)) = 1 \qquad [3.3]$$

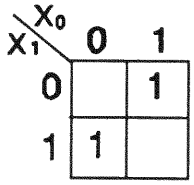
And we normalize μ_U in the interval [0 1].

3.4 Determination of ρ_n

3.4.1 For two variables (n=2)

Example 3.3. The Exclusive OR function

$$f_{XOR} = \Sigma(1,2)$$



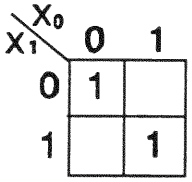
$f_{XOR} = x_0 \oplus x_1 = x_0'x_1 + x_0x_1'$. From definition 3.3, if there is no variable appears

both uncomplemented and complemented when f is written in a minimum sum of products form, then function $f(x_1,x_2,...,x_n)$ is said to be unate in x_i and f is a unate function. Now variables x_0 and x_1 appear both uncomplemented and complemented. So this is not a unate function and the

minimum number of minterms to be changed is equal to one in order to change the function f_{XOR} to be a unate function. $C(f_{XOR}) = 1$.

Example 3.4. The Exclusive NOR function

$$f_{EQU} = \Sigma(0,3)$$



$f_{EQU} = x_0 \odot x_1 = x_0x_1 + x_0'x_1'$. Using the same definition, variables x_0 and x_1 appear both uncomplemented and complemented now. So this is not a unate function and the minimum number of minterms to be changed is equal to one in order to change the function f_{EQU} to be a unate function. $C(f_{EQU}) = 1$.

From example 3.1 and 3.2, we can conclude the following theorems.

THEOREM 3.1. For all functions f of two variables, the max value $C(f(x_1,x_2))$ is equal to 1.

$$C_{max}(2) = \max_{f}\{C_U(f(x_1,x_2))\} = 1 \qquad [3.4]$$

THEOREM 3.2. For all functions f of two variables, the normalization constant ρ_2 can be obtained by as follows.

$$\mu_{MNU}(f(x_1, x_2)) = 1-\rho_2 C_{\max}(2) \quad [3.5]$$

$$0 = 1-\rho_2 C_{\max}(2) \quad [3.6]$$

$$\rho_2 = \frac{1}{C_{\max}(2)} = 1 \quad [3.7]$$

THEOREM 3.3. For all functions f of two variables, the grade of membership function is:

$$\mu_U(f(x_1, x_2)) = 1-\frac{C}{C_{\max}(2)} = 1-C \quad [3.8]$$

3.4.2 For three variables (n=3)

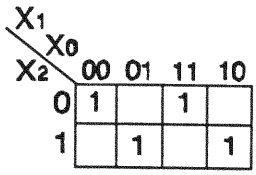
Example 3.5. $f_{odd}(3) = \Sigma(1,2,4,7)$

X_2 \ X_1X_0	00	01	11	10
0		1		1
1	1		1	

$f_{odd}(3) = x_0 \oplus x_1 \oplus x_2 = x_0x_1'x_2' + x_0'x_1x_2' + x_0'x_1'x_2 + x_0x_1x_2$. Variables x_0, x_1, and x_2 all appear uncomplemented and complemented now. So $f_{odd}(3)$ is not a unate function and the minimum number of minterms to be changed is equal to one in order to change it to

be a unate function. $C(f_{odd}(3)) = 1$.

Example 3.6. $f_{even}(3) = \Sigma(0,3,5,6)$



$f_{even}(3) = x_0 \odot x_1 \odot x_2 = x_0'x_1'x_2' + x_0x_1x_2' + x_0x_1'x_2 + x_0'x_1x_2$. Variables x_0, x_1, and x_2 all appear uncomplemented and complemented now. So this function is not a unate function and the minimum number of minterms to be changed is equal to one in order to change $f_{even}(3)$ to be a unate function. $C(f_{even}(3)) = 1$.

From above, we can conclude all three-variable functions have the following properties.

THEOREM 3.4. For all functions f of three variables, the normalization constant ρ_3 can be obtained by as follows.

$$C_{max}(3) = 2 \qquad [3.9]$$

$$\mu_{MNU}(f(x_1, x_2, x_3)) = 1 - \rho_3 C_{max}(3) \qquad [3.10]$$

$$0 = 1 - \rho_3 C_{max}(3) \qquad [3.11]$$

$$\rho_3 = \frac{1}{C_{max}(3)} = \frac{1}{2} \qquad [3.12]$$

THEOREM 3.5. For all function of three variables, the grade membership function is:

$$\mu_U(f(x_1, x_2, x_3)) = 1 - \frac{C}{C_{max}(3)} = 1 - \frac{C}{2} \qquad [3.13]$$

3.4.3 For n variables

From the discussions of a function with two variables and three variables, the grade of membership function of a n-variables function can be generalized as:

$$\mu_U(f(x_1, x_2, \ldots, x_n)) = 1 - \frac{C}{C_{max}(n)} \qquad [3.14]$$

3.4.4 Summary

From above theorems and examples, we can summarize the results as following theorems, and tables. Furthermore, the algorithm for determining $C_{max}(n)$ and C(n), and the algorithm for generating most non-unate functions are also can be achieved by using these results.

In chapter 1, we discussed fuzzy threshold functions and defined most inseparable functions are equal to odd function and even function. From the above derivation of two and three variables, we know that most non-unate functions are similar to threshold functions in fuzzy unate functions. They are also odd function and even function.

DEFINITION 3.5. Thus, we define the functions are most dissimilar unate functions to be the most non-unate functions f_{MNU} which are odd function and even function. And then we can determine the maximum number of minterms change C_{max} and normalization constant ρ_n of n variables.

$$C_{MNU}(f(x_1, x_2, \ldots, x_n)) = \max_{\forall f}\{C_U(f(x_1, x_2, \ldots, x_n)\} \qquad [3.15]$$

From definition 3.5, it indicated that the two most non-unate functions are as shown in Figure 3.1.

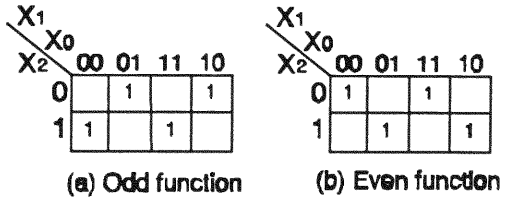


Figure 3.1. A map for two most non-unate functions of three variables

For any two and three-variable functions, we can summarize the results as table 3.1.

No. of variables	C_{max}	No. of most non-unate functions	most non-unate functions	membership function	possible values of μ_U
2	1	2	$\Sigma(0,3)$ $\Sigma(1,2)$	$\mu_U(f)$ = 1-C	0,1
3	2	2	$\Sigma(0,3,5,6)$ $\Sigma(1,2,4,7)$	$\mu_U(f)$ = 1-(C/2)	0,1/2,1

Table 3.1. Summary of two and three-variable functions

3.5 An algorithm for determining $C_{max}(n)$

THEOREM 3.6. For all functions of four variables, the maximum minterms change $C_{max}(4)$ in order to convert f to be a unate function is equal to 7.

THEOREM 3.7. For all functions of five variables, the maximum minterms change $C_{max}(5)$ in order to convert f to be a unate function is equal to 15.

THEOREM 3.8. For all functions of n variables with n≥4, the maximum number of minterms changes is:

$$C_{max}(n) = 2^{n-1}-1 \qquad [3.16]$$

So the results can be summarized as shown in Table 3.2.

number of variables	C_{max}
4	7
5	15
6	31
7	63
. . .	. . .
n	$2^{n-1}-1$

Table 3.2. Summary of $C_{max}(n)$ of n variables

THEOREM 3.9. For all member of variables, the number of most non-unate functions are always equal to 2.

THEOREM 3.10. f is a unate function if and only if f' is a unate function. This theorem can be generalized to fuzzy unate functions.

THEOREM 3.11. For all functions f of n variables, then

$$C(f) = C(f') \qquad [3.17]$$

$$\mu_U(f) = \mu_U(f') \qquad [3.18]$$

3.6 An algorithm for determining C(n)

In this section, we can also use Boolean lattice representation (chapter 1) to help us determining a function is whether a unate function or not. So I would like to explain the Boolean lattice representation again because we will use some terminologies here.

An *n-variable Boolean lattice representation* contains 2^n vertices, each of which represents an assignment of values to the n variables and thus corresponds to a minterm. A line is drawn between every pair of vertices which differ in just one variable, and no other lines are drawn. The vertices corresponding to true minterms, that is, for which the function assumes the value logic one, are called *true vertices*, while those corresponding to false minterms are called *false vertices.*

THEOREM 3.12. [21] A logic function $f(x_1,x_2,...,x_n)$ is unate if and only if it is not tautology and partial ordering exists, so that for every pair of vertices $(a_1,a_2,...,a_n)$ and $(b_1,b_2,...,b_n)$, if $(a_1,a_2,...,a_n)$ is a true vertex and $(b_1,b_2,...,b_n) \geq (a_1,a_2,...,a_n)$, then $(b_1,b_2,...,b_n)$ is also a true vertex of f.

Note. Partial ordering set of vertices is a lattice and the vertices (0,0,...,0) and (1,1,...,1) are, respectively, *the least vertex* and *the greatest vertex* of the lattice.

According to theorem 3.12, a procedure can be written in order to change any n-variable function into a unate function and to decide the minimum number of minterms change.

Step 1. To find all the subcubes' minterms and minimal true vertices of each subcube.

Step 2. To check partial ordering relation of every subcube.

If yes, the function is a unate function.

If not, change the minterms which does not satisfy the partial ordering relation.

Step 3. To determine the number of minterms change.

Example 3.7. For a function $f_1(x_1,x_2,x_3) = x_1'x_2'+x_1x_3$ and the three-variable Boolean lattice representation is shown in Figure 3.2.

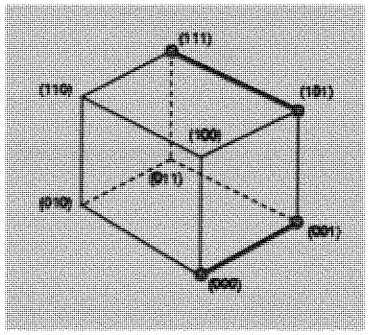


Figure 3.2. The Boolean lattice representation of $f_1(x_1,x_2,x_3) = x_1'x_2'+x_1x_3$

Step 1. There are two pairs of vertices [(0,0,1) and (0,0,0), (1,1,1) and (1,0,1)] and we name the two minimal true vertices are S_1=(0,0,0) and S_2=(1,0,1).

Step 2. According to partition ordering, once the least vertex (0,0,0) is logic one, then all the others vertices should be true vertices. But the vertices (1,0,0), (0,1,0), (0,1,1), and (1,1,0) are not true vertices now. So we change the least vertex (0,0,0) to false vertex and minterm (0,0,1) becomes as a minimal true vertex.

Step 3. To check the partial ordering relation of these two subcubes. We found (0,1,1)

is not a true vertex which true vertex (0,0,1) is not covered by it. So we have to change (0,1,1) to be a true vertex.

Step 4. From the above steps, we found that we changed two vertices, (0,0,0) and (0,1,1), and f_1 becomes as a unate function -- $f_1(x_1,x_2,x_3) = x_1'x_3+x_1x_3 = x_3$ and C(n) is equal to 2. The Boolean lattice representation is shown in Figure 3.3.

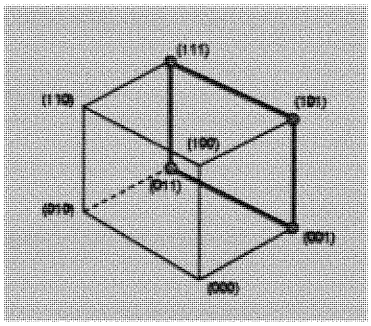


Figure 3.3. The Boolean lattice representation of $f_1(x_1,x_2,x_3) = x_1'x_3'+x_1x_3 = x_3$

Example 3.8. For a function $f_2(x_1,x_2,x_3,x_4) = x_1x_2+x_3x_4$ and the four variables Boolean lattice representation is shown in Figure 3.4.

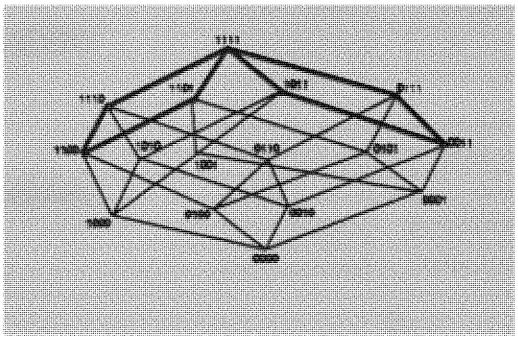


Figure 3.3. The Boolean lattice representation of $f_2(x_1,x_2,x_3,x_4) = x_1x_2+x_3x_4$

Step 1. There are two pairs of vertices [(1,1,0,0), (1,1,1,0), (1,1,0,1), and (1,1,1,1), (0,0,1,1), (1,0,1,1), (0,1,1,1), and (1,1,1,1)] and we name the two minimal true vertices are S_1=(0,0,1,1) and S_2=(1,1,0,0).

Step 2. To check the partial ordering relation of these two subcubes. We found that every vertex are greater than S_1 and S_2.

Step 3. From the above steps, we can say that f_2 is a unate function and the number of minterms change C(n) is equal to 0.

3.7 An algorithm for generating most non-unate functions f_U

When we have most non-unate functions of two variables, we can generate most non-unate functions of three, four, five variables, etc. Figure 3.5 shows the algorithm for generating most non-unate functions f_{MNU}.

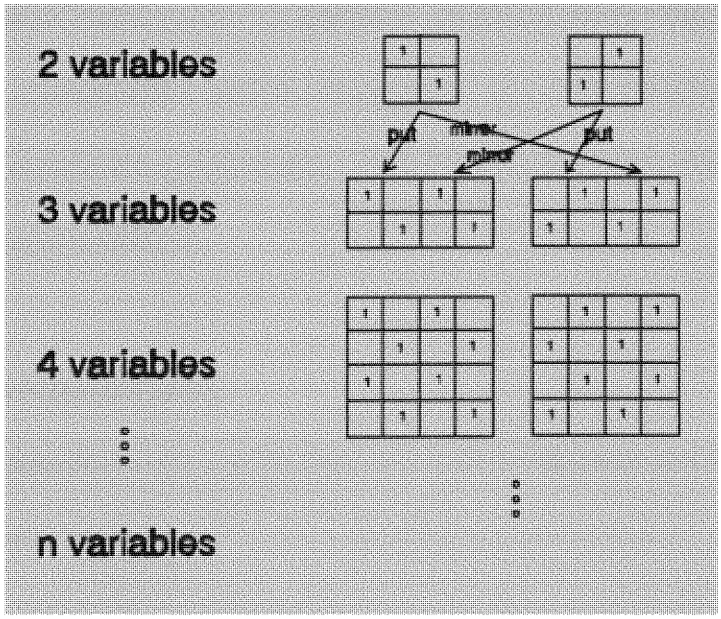


Figure 3.5. The algorithm for generating most non-unate functions f_{MNU}

THEOREM 3.13. For n variables, there are only two most non-unate functions (f_U) and denoted as $f_{even}(n)$ and $f_{odd}(n)$. Their relationship is:

$$f_{even}(n) = f'_{odd}(n) \qquad [3.19]$$

THEOREM 3.14. Given $f_{even}(n)$ and $f_{odd}(n)$, $f_{even}(n+1)$ and $f_{odd}(n+1)$ can be generated as follows.

$$f_{even}(n+1) = x'_{n+1} f_{even}(n) + x_{n+1} f_{odd}(n) \qquad [3.20]$$

$$f_{odd}(n+1) = x_{n+1} f_{even}(n) + x'_{n+1} f_{odd}(n) \qquad [3.21]$$

3.8 Applications

Fuzzy unate functions can be applied to switching theory, logic design, computer design, pattern classification, pattern recognition, and pattern understanding. In this chapter, I discuss their three applications -- function representation, data compression, and error detection and correction.

3.8.1 Function representation

Fuzzy unate functions can be used as a tool to represent any logic function f.

If C(n) = i, then

$$f = f_U \oplus \Sigma(m_1, m_2, \ldots, m_i) \qquad [3.22]$$

$$f_U = f \oplus \Sigma(m_1, m_2, \ldots, m_i) \qquad [3.23]$$

where f_U is a unate function, $\oplus$ indicates the exclusive or operation and m_1, m_2,..., m_i are minterms which should be changed from f_U in order to obtain f.

Example 3.9. If we want to convert $f(x_1,x_2,x_3) = \Sigma(1,2,4,7)$ into a unate function $f_U(x_1,x_2,x_3)$, then $C(f(x_1,x_2,x_3))=1$ and it can be represented as $f(x_1,x_2,x_3) = \Sigma(1,2,3,7)\oplus\Sigma(m_3,m_4)$.

3.8.2 Data compression

A unate function can be represented $[w_1,w_2,...,w_n;C]$ which has a simpler representation compare with a general logic function.

For function f with C(n)=i, then function f can be represented by

$$f = f_U \oplus \{m_i\} \qquad [3.24]$$

Thus, for transmitting f, instead of transmitting the total truth table or the corresponding Karnaugh map, we can transmit only f_U and the minterms m_i. Therefore, data compression may be achieved.

3.8.3 Error detection and correction

Fuzzy unate functions also can be applied to error detection. For example, if we know that a most non-unate function (f_{MNU}) is transmitted and errors occurred, than at the receiving end,

$$C(f'_{MNU}) \neq C_{max} \qquad [3.25]$$

where f'_{MNU} indicates the function received.

We know that there are only two most non-unate functions for n variables. Therefore, if we transmit a most non-unate function of n variables, then all the errors can be detected and corrected.

Example 3.10. $f_{MNU}(x_1,x_2,x_3) = \Sigma(0,3,5,6;C=2)$ is transmitted, $f(x_1,x_2,x_3) = \Sigma(0,1,3,5;C=0)$ is received. $C(f_{MNU}(x_1,x_2,x_3))=2$ is not equal to $C(f(x_1,x_2,x_3))=0$. So the numbers of minterms 1 and 5 of f are erroneous and they can be corrected.

3.9 Conclusion

Fuzzy unate functions are introduced and investigated by using the concept of fuzzy logic. We know that there are two most non-unate functions and they are same as the most inseparable functions of threshold functions, they are all odd function and even function. The algorithms for determining C(n) and $C_{max}(n)$, generating most non-unate functions f_U are presented and illustrated with examples. About applications, fuzzy unate functions are also applied to function representation, data compression, and error detection and correction. The results may be have useful applications in logic design, pattern recognition, and related areas.

Chapter Four

Flash Analog to Digital Converter

4.1 Introduction

When an analog voltage must be converted to a digital number, an analog to digital converter (A/D converter) is used. Figure 4.1 shows a block diagram symbol for a typical A/D converter with a single analog input and several digital outputs.

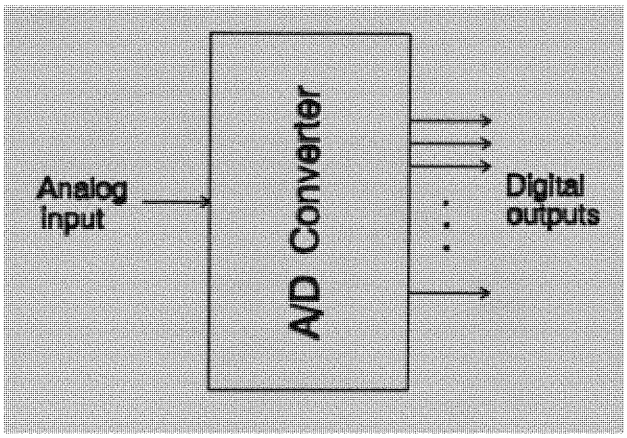


Figure 4.1. Block diagram of a typical A/D Converter

There are many types of A/D converters, most of which are single slope A/D converters, dual slope A/D converters, successive approximation A/D converters, tracking A/D converters, counter ramp converters, flash A/D converters, and so on. Since the flash A/D converter is the fastest converter and all comparators outputs are monotone increasing (decreasing) functions, I concentrate my discussion on this subject in the following sections.

4.2 Basic structure

All parallel, or so called *simultaneous*, or *flash* converters offer the fastest throughout of available A/D converters designs. All the bit choices of the converter are made at the same time. It requires 2^n-1 voltage-divider taps and comparators -- and a comparable amount of priority encoder logic. A scheme that gives extremely fast conversion, it requires large numbers of nearly identical components, hence it is well-suited to and really feasible in integrated circuit form. Figure 4.2 illustrates a typical block diagram [1,2,8,20].

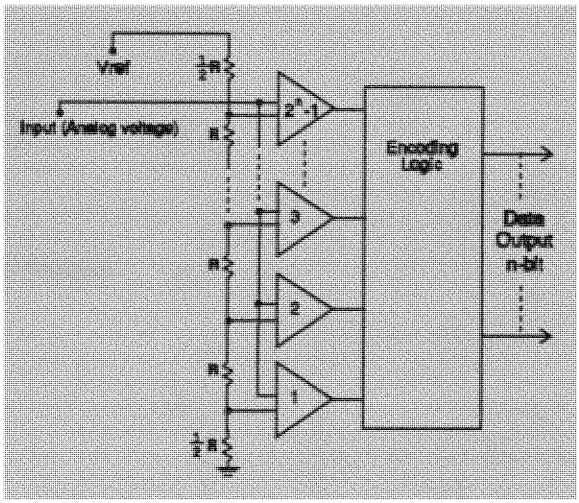


Figure 4.2. flash A/D converter

When an analog input signal is presented at the input of the comparator bank, all comparators which have a reference voltage below the level of the input signal will assume a logic one output. The comparators which have their reference voltage above the input signal will have a logic zero outputs. Encoding logic then converts the comparators' outputs into a binary digital output.

4.3 2-bit flash analog to digital converter

A 2-bit flash analog to digital converter consists of one reference voltage, three comparators, three flip-flops and one priority encoder.

Figure 4.3 shows the diagram of a 2-bit flash analog to digital converter. The relationship between outputs C_1, C_2, and C_3 from comparators and the outputs Z_1 and Z_2 from priority encoder is shown in Table 4.1.

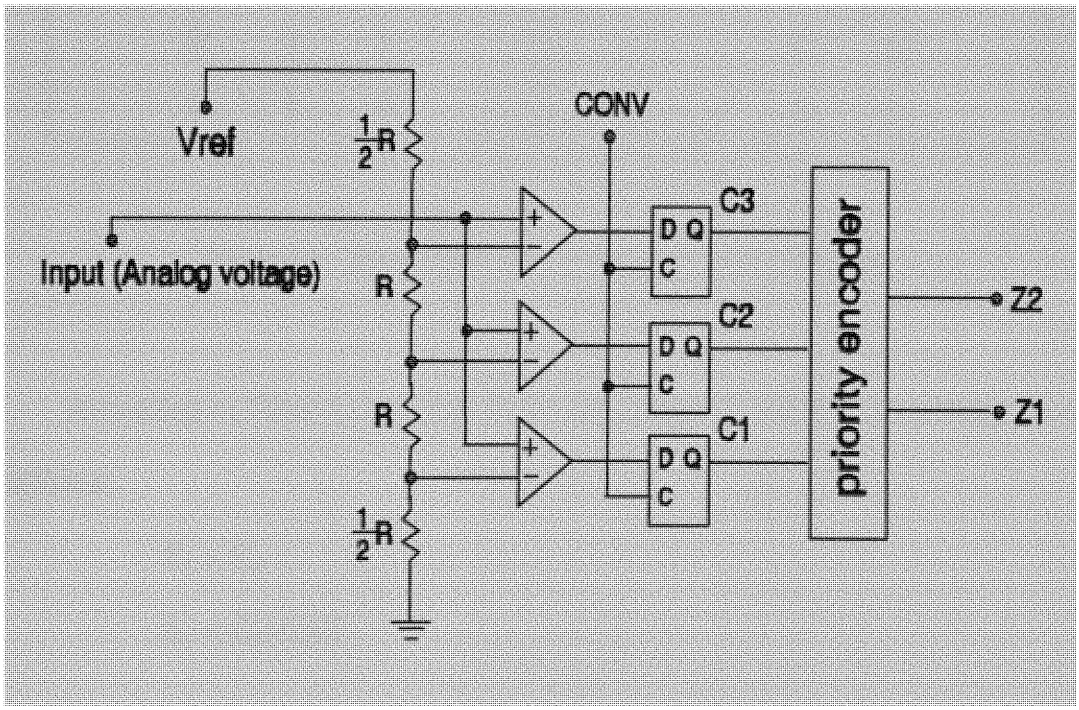


Figure 4.3. A high-speed 2-bit flash A/D converter

2-bit flash A/D converter				
Outputs from comparators			Outputs from priority encoder	
C3	C2	C1	Z2	Z1
0	0	0	0	0
0	0	1	0	1
0	1	1	1	0
1	1	1	1	1

Table 4.1. Truth table of a 2-bit flash A/D converter

According to Table 4.1, we can derive the outputs Z_1 and Z_2 from priority encoder as follows:

$$Z_2 = C_2$$

$$Z_1 = C_3 + C_1 C_2'$$

4.4 3-bit flash analog to digital converter

A 3-bit flash analog to digital converter consists of one reference voltage, seven comparators, seven flip-flops and one priority encoder.

Figure 4.4 shows the diagram of a 3-bit flash analog to digital converter. The relationship between outputs C_1, C_2, C_3, C_4, C_5, C_6, and C_7 from comparators and the outputs Z_1, Z_2, and Z_3 from priority encoder is shown in Table 4.2.

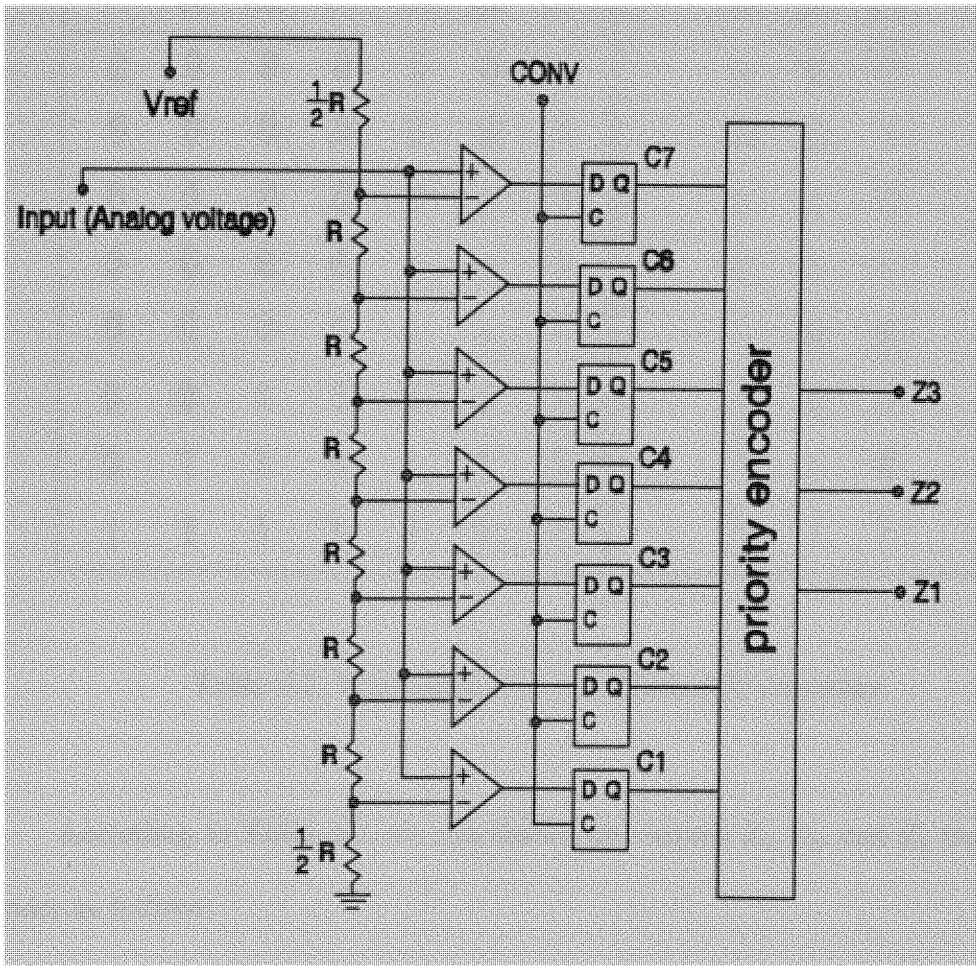


Figure 4.4. A high-speed 3-bit flash A/D converter

3-bit flash A/D converter									
Outputs from comparators							Outputs from priority encoder		
C7	C6	C5	C4	C3	C2	C1	Z3	Z2	Z1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	1	0	1	0
0	0	0	0	1	1	1	0	1	1
0	0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1

Table 4.2. Truth table of a 3-bit flash A/D converter

According to Table 4.2, we can derive the outputs Z_1, Z_2, and Z_3 from priority encoder as follows:

$$Z_3 = C_4$$

$$Z_2 = C_6 + C_2C_4'$$

$$Z_1 = C_7 + C_5C_6' + C_3C_4' + C_1C_2'$$

4.5 4-bit flash Analog to digital converter

A 4-bit flash analog to digital converter consists of one reference voltage, fifteen comparators, fifteen flip-flops and one priority encoder.

Figure 4.5 shows the diagram of a 4-bit flash analog to digital converter. The relationship between outputs C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, and C_{15} from comparators and the outputs Z_1, Z_2, Z_3 and Z_4 from priority encoder is shown in Table 4.3.

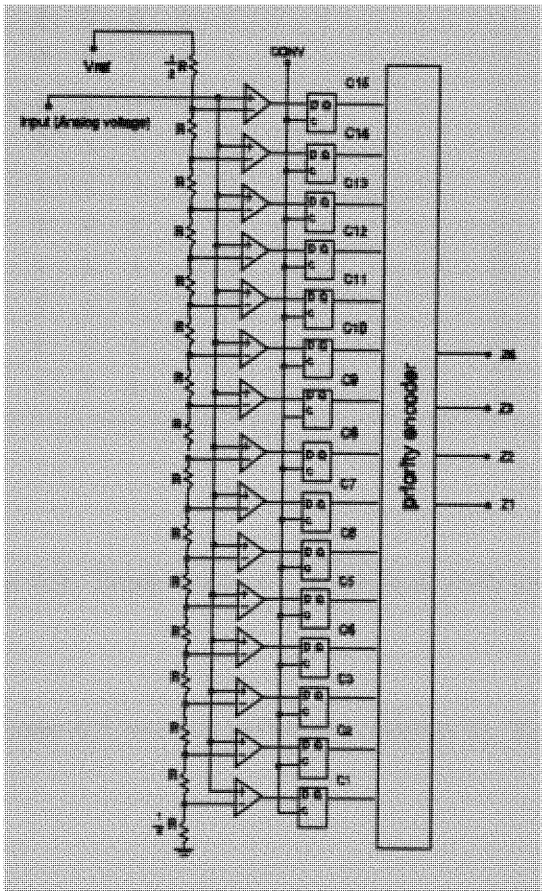


Figure 4.5. A high-speed 4-bit flash A/D converter

4-bit flash A/D converter																	
Outputs from comparators															Outputs from priority encoder			
C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	Z4	Z3	Z2	Z1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 4.3. Truth table of a 4-bit flash A/D converter

According to Table 4.3, we can derive the outputs Z_1, Z_2, Z_3, and Z_4 from priority encoder as follows:

$$Z_4 = C_8$$

$$Z_3 = C_{12}+C_4C_8'$$

$$Z_2 = C_{14}+C_{10}C_{12}'+C_6C_8'+C_2C_4'$$

$$Z_1 = C_{15}+C_{13}C_{14}'+C_{11}C_{12}'+C_9C_{10}'+C_7C_8+C_5C_6'+C_3C_4'+C_1C_2'$$

4.6 Conclusion

According to the derivations of 2-bit, 3-bit, and 4-bit flash A/D converters, we can generalize the requirements of a n-bit A/D flash converter and the outputs' Boolean functions from priority encoder.

(1) the requirements of a n-bit A/D flash converter are:

the number of reference voltage: 1

the number of comparators: 2^n-1

the number of flip-flops: 2^n-1

the number of priority encoder: 1

(2) the outputs' Boolean functions from priority encoder are:

$$Z_n = C_{2^n/2} \qquad [4.1]$$

$$Z_{n-1} = C_{3*2^n/4} + C_{1*2^n/4}C'_{2*2^n/4} \qquad [4.2]$$

$$Z_{n-2} = C_{7*2^n/8} + C_{5*2^n/8}C'_{6*2^n/8} + C_{3*2^n/8}C'_{4*2^n/8} + C_{1*2^n/8}C'_{2*2^n/8} \qquad [4.3]$$

$$\begin{aligned} Z_{n-3} = {} & C_{15*2^n/16} + C_{13*2^n/16}C'_{14*2^n/16} + C_{11*2^n/16}C'_{12*2^n/16} + C_{9*2^n/16}C'_{10*2^n/16} \\ & + C_{7*2^n/16}C'_{8*2^n/16} + C_{5*2^n/16}C'_{6*2^n/16} + C_{3*2^n/16}C'_{4*2^n/16} + C_{1*2^n/16}C'_{2*2^n/16} \end{aligned} \qquad [4.4]$$

.

.

.

Chapter Five

Minimum Universal Synthesis of Logic Functions by Using Monotone Flash Analog to Digital Converters

5.1 Introduction

In chapter 2, I mentioned that all the outputs from comparators of a n bit flash analog to digital converter are a set of monotone increasing (decreasing) functions. Also, through the discussion of chapter 4, it is known possible to synthesize these n binary outputs from priority encoder by using the combination of outputs from 2^n-1 comparators of a n-bit flash analog to digital converter. I concentrated the discussion only on synthesizing these outputs from priority encoder. So I want to extend the discussion more deeply about the synthesizing rules to any given function by using monotone flash analog to digital converters in this chapter.

5.2 Rule of choosing minterms or maxterms to synthesize a given function

If a logic function is given, there must be a certain number of minterms and maxterms. The following Lemmas will decide how to choose either minterms or maxterms to synthesize a logic function.

LEMMA 5.1. For any given function of n variables, the absolute distance between the number of segment of logic one and the number of segment of logic zero should be equal to or smaller than one.

$$| \#\ of\ segment\ of\ 1 - \#\ of\ segment\ of\ 0 | \le 1 \qquad [5.1]$$

IF # of segment of 1 > # of segment of 0, THEN

Using logic zero (maxterm) to synthesize the function.

IF # of segment of 0 > # of segment of 1, THEN

Using logic one (minterm) to synthesize the function.

Thus, Lemma 5.1 can be transferred into Figure 5.1.

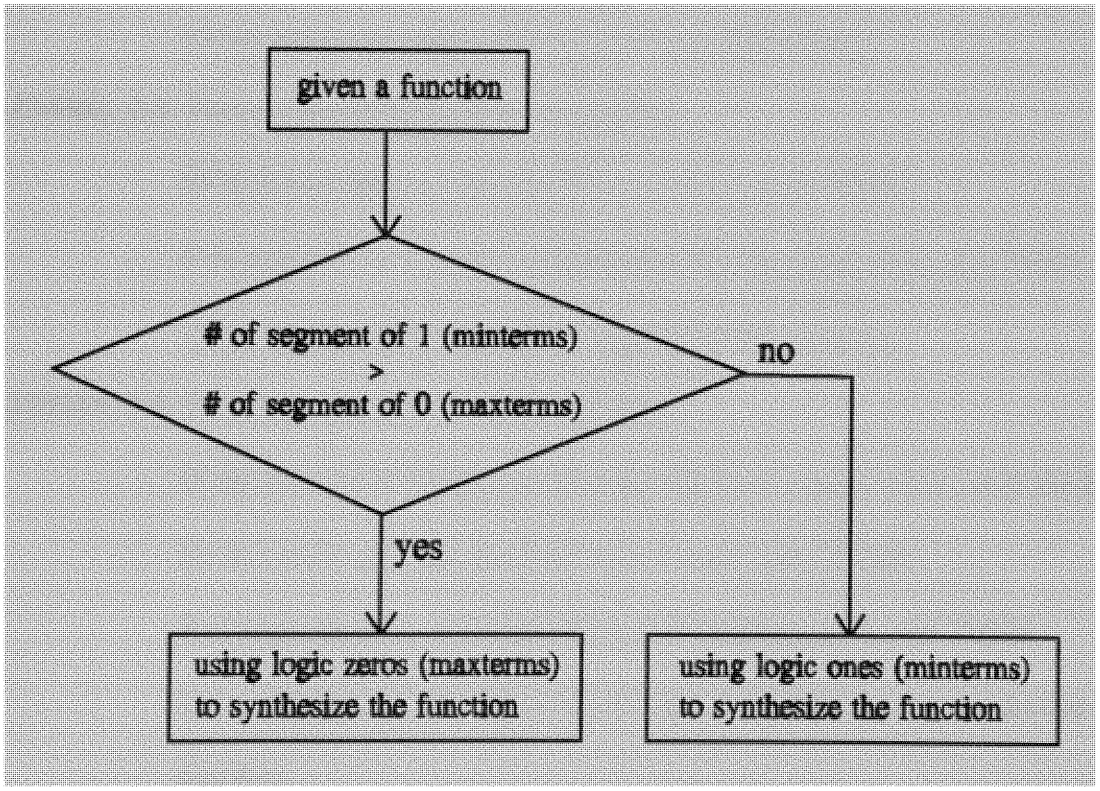


Figure 5.1. The first rule for choosing minterms or maxterms to synthesize a given function

Example 5.1. For a given three-variable function $f(x_1,x_2,x_3)=\Sigma(1,2,6)$, the # of segment of 1 which is equal to two is less than the # of segment of 0 which is equal to three. As mentioned above, it is better to use logic one (minterm) to synthesize this function.

LEMMA 5.2. For any given function of n variables,

IF both the first and the last bits are logic zeros, THEN

# of segment of 0 will be greater than # of segment of 1, and THEN

Using logic one (minterm) to synthesize the function.

LEMMA 5.3. For any given function of n variables,

IF both the first and the last bits are logic ones, THEN

# of segment of 1 will be greater than # of segment of 0, and THEN

Using logic zero (maxterm) to synthesize the function.

LEMMA 5.4. For any given function of n variables,

IF the first bit is a logic one and the last bit is a logic zero , or if the first bit is a logic zero and the last bit is a logic one, THEN

# of segment of 1 will be equal to # of segment of 0, and THEN

Using either logic one (minterm) or logic zero (maxterm) to synthesize the function.

And through Lemma 5.2 to 5.4 can be summarized as Figure 5.2.

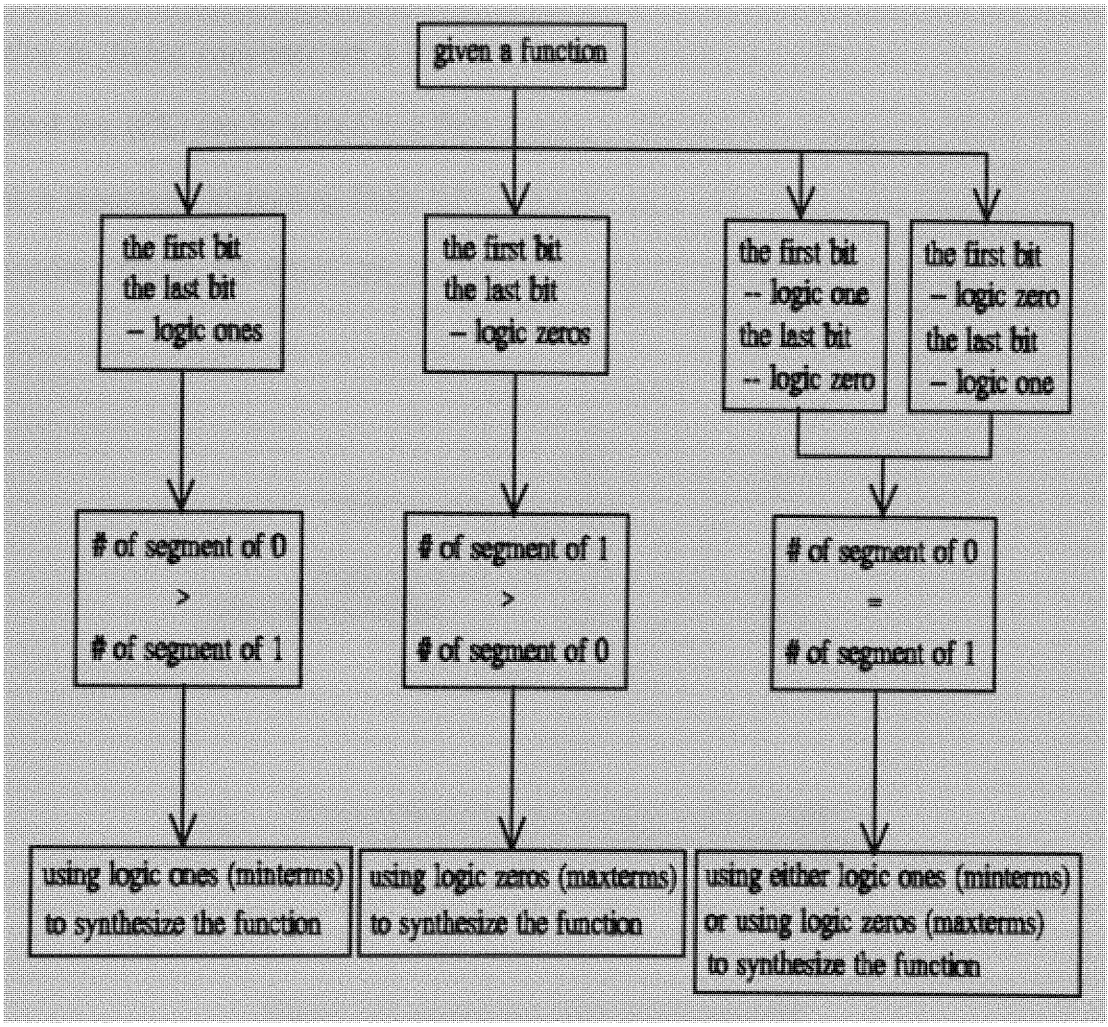


Figure 5.2. The second rule for choosing minterms or maxterms to synthesize a given function

Example 5.2. For a given function $f(x_1,x_2,x_3)=\Sigma(0,3,4,5,7)$ of three variables, both the first bit and the last bit are logic ones. According to Lemma 5.3, it will be more effective to use logic zero (maxterm) to synthesize this function.

5.3 Synthesis of a logic function by using monotone flash analog to digital converters

In chapter 2, I mentioned that all the outputs from comparators of a n-bit flash analog to digital converter are a set of monotone increasing (decreasing) functions. A control line (CL) can be added to a flash analog to digital converter in order to converter it as a monotone increasing flash A/D converter or a monotone decreasing flash A/D converter. Figure 5.3 shows the diagram of a monotone flash A/D converter.

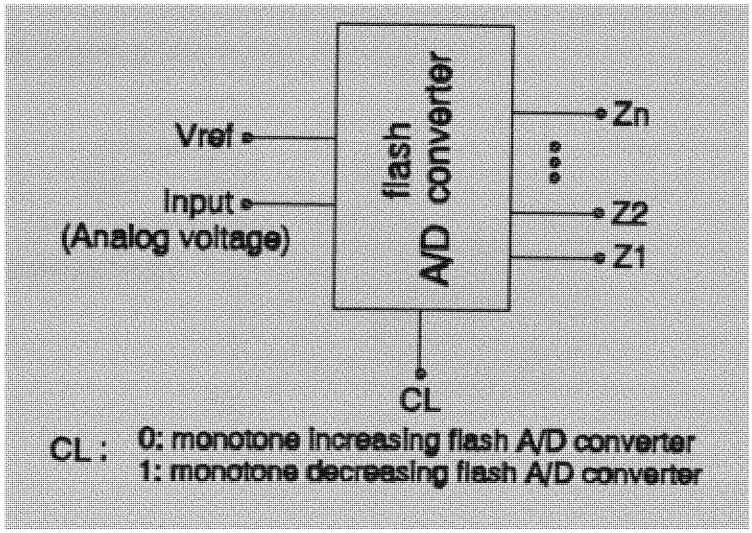


Figure 5.3. monotone flash A/D converter

When CL is set to 0, the converter will become a monotone increasing flash A/D converter.

When CL is set to 1, the converter will become a monotone decreasing flash A/D converter.

5.3.1 Synthesis of a logic function by using monotone increasing flash analog to digital converters

Figure 5.4 shows an actual n-bit monotone increasing flash analog to digital converter which is designed by using a set of monotone increasing functions. To analysis the choice of using either logic one or logic zero to synthesize a given logic function, Lemma 5.1 through 5.4 are applied to any given logic function. If the given function fits to Lemma 5.2, Table 5.1a is used to synthesize the function, whereas Table 6.1b is used to synthesize the function which fits to Lemma 5.3.

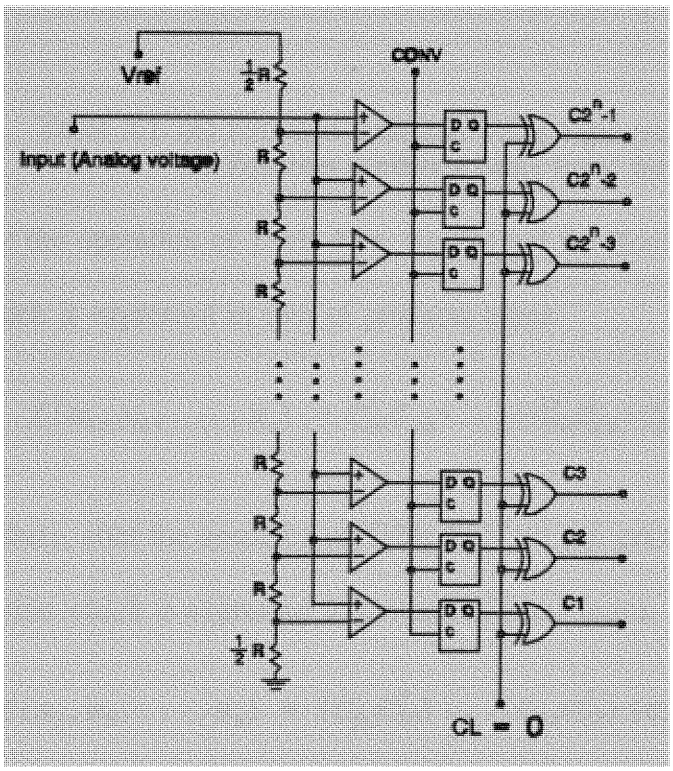


Figure 5.4. A n-bit flash analog to digital converter which is designed by using a set of monotone increasing functions

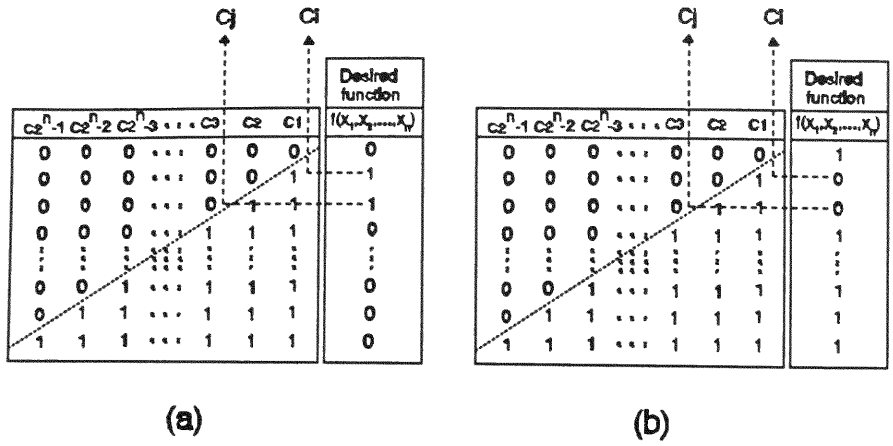


Table 5.1. (a) Function synthesis using logic one (minterm)
(b) Function synthesis using logic zero (maxterm)

After the operation of Table 5.1a, we can get the Boolean expression of a logic function which is expressed in sum of products terms.

$$f(x_1, x_2, \ldots, x_n) = C'_{jn}C_{in} + \ldots + C'_{j2}C_{i2} + C'_{j1}C_{i1} \qquad [5.2]$$

Similarly, after the operation of Table 5.1b, we can get the Boolean expression of a logic function which is expressed in product of sums terms.

$$f(x_1, x_2, \ldots, x_n) = (C_{jn} C'_{in}) \ldots (C_{j2} C'_{i2})\ (C_{j1} C'_{i1}) \qquad [5.3]$$

It was obviously that we can synthesize any function by using either Table 5.1a or Table 5.1b. As follows, I illustrate several examples to explain how these two methods really work.

Example 5.3. Minimum synthesis of three variables function $f(x_1,x_2,x_3)=\Sigma(2,3,4,5,6)$ by using a 3-bit monotone increasing flash analog to digital converter in Table 5.2.

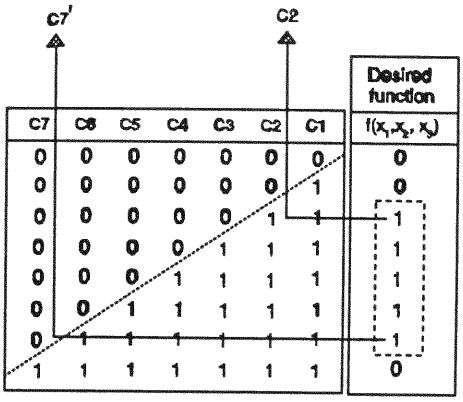


Table 5.2. Minimum synthesis of $f(x_1,x_2,x_3)=\Sigma(2,3,4,5,6)$ by using a 3-bit monotone increasing flash A/D converter

In this example, since the # of segment of 0 which is equal to two is greater than the # of segment of 1 which is equal to one and both the first and the last bits are logic zeros, we apply Table 5.1a and equation 5.2. Then we can create the truth table 5.3.

C_7'	C_2	$f(x_1,x_2,x_3) = C_7'C_2$
1	0	0
1	0	0
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
0	1	0

Table 5.3

As a result, the minimum function $f(x_1,x_2,x_3) = C_7'C_2$.

Example 5.4. Minimum synthesis of three variables function $f(x_1,x_2,x_3)=\Sigma(1,2,6)$ by using a 3-bit monotone increasing flash analog to digital converter in Table 5.4.

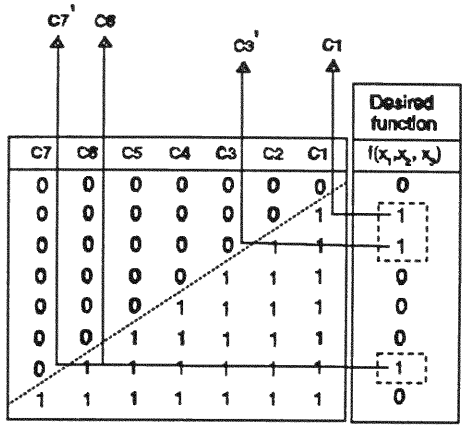


Table 5.4. Minimum synthesis of $f(x_1,x_2,x_3)=\Sigma(1,2,6)$ by using a 3-bit monotone increasing flash A/D converter

In this example, since the # of segment of 0 which is equal to three is greater than the # of segment of 1 which is equal to two and both the first and the last bits are logic zeros, we apply Table 5.1a and equation 5.2. We can then create the truth table 5.5.

C_7'	C_6	$C_7'C_6$	C_3'	C_1	$C_3'C_1$	$f(x_1,x_2,x_3) = C_7'C_6+C_3'C_1$
1	0	0	1	0	0	0
1	0	0	1	1	1	1
1	0	0	1	1	1	1
1	0	0	0	1	0	0
1	0	0	0	1	0	0
1	0	0	0	1	0	0
1	1	1	0	1	0	1
0	1	0	0	1	0	0

Table 5.5

As a result, the minimum function $f(x_1,x_2,x_3) = C_7'C_6+C_3'C_1$.

Example 5.5. Minimum synthesis of three variables function $f(x_1,x_2,x_3)=\Sigma(0,1,2,6,7)$ by using a 3-bit monotone increasing flash analog to digital converter in Table 5.6.

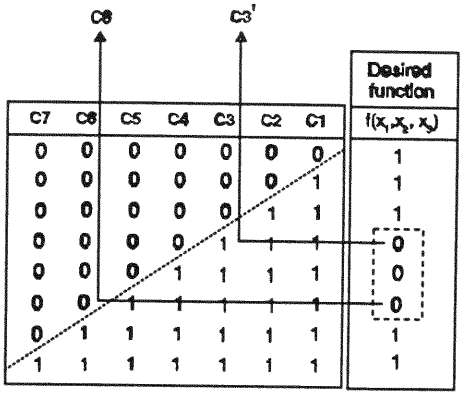


Table 5.6. Minimum synthesis of $f(x_1,x_2,x_3)=\Sigma(0,1,2,6,7)$ by using a 3-bit monotone increasing flash A/D converter

In this example, since the # of segment of 0 which is equal to one is less than the # of segment of 1 which is equal to two and both the first and the last bits are logic ones, we apply Table 5.1b and equation 5.3. Then we can create the truth table 5.7.

C_6	C_3'	$f(x_1,x_2,x_3) = C_6+C_3$'
0	1	1
0	1	1
0	1	1
0	0	0
0	0	0
0	0	0
1	0	1
1	0	1

Table 5.7

As a result, the minimum function $f(x_1,x_2,x_3) = C_6+C_3$'.

Example 5.6. Minimum synthesis of three variables function $f(x_1,x_2,x_3)=\Sigma(0,4,5,7)$ by using a 3-bit monotone increasing flash analog to digital converter in Table 5.8.

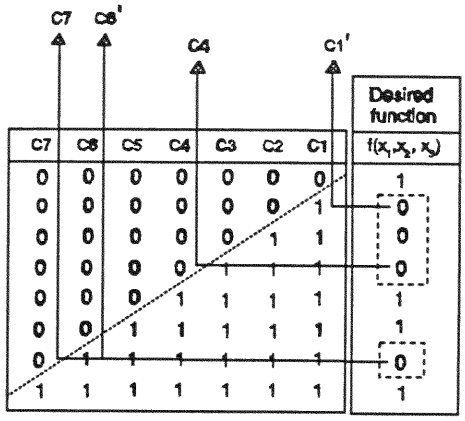


Table 5.8. Minimum synthesis of $f(x_1,x_2,x_3)=\Sigma(0,4,5,7)$ by using a 3-bit monotone increasing flash A/D converter

In this example, since the # of segment of 0 which is equal to two is less than the # of segment of 1 which is equal to three and both the first and the last bits are logic ones, we apply Table 5.1b and equation 5.3. Then we can create the truth table 5.9.

C_7 C_6'	C_7+C_6'	C_4 C_1'	C_4+C_1'	$f(x_1,x_2,x_3) = (C_7+C_6')(C_4+C_1')$
0 1	1	0 1	1	1
0 1	1	0 0	0	0
0 1	1	0 0	0	0
0 1	1	0 0	0	0
0 1	1	1 0	1	1
0 1	1	1 0	1	1
0 0	0	1 0	1	0
1 0	1	1 0	1	1

Table 5.9

As a result, the minimum function $f(x_1,x_2,x_3) = (C_7+C_6')(C_4+C_1')$.

5.3.2 Synthesis of a logic function by using monotone decreasing flash analog to digital converters

Figure 5.5 shows an n-bit monotone decreasing flash analog to digital converter which is designed by using a set of monotone decreasing functions. To analysis the choice of using either logic one or logic zero to synthesize a given logic function, we apply the same strategy of section 5.3.1 to the given function. If the given function fits to Lemma 5.2, Table 5.10a is used to synthesize the function. And Table 5.10b is used to synthesize the given function which fits to Lemma 5.3.

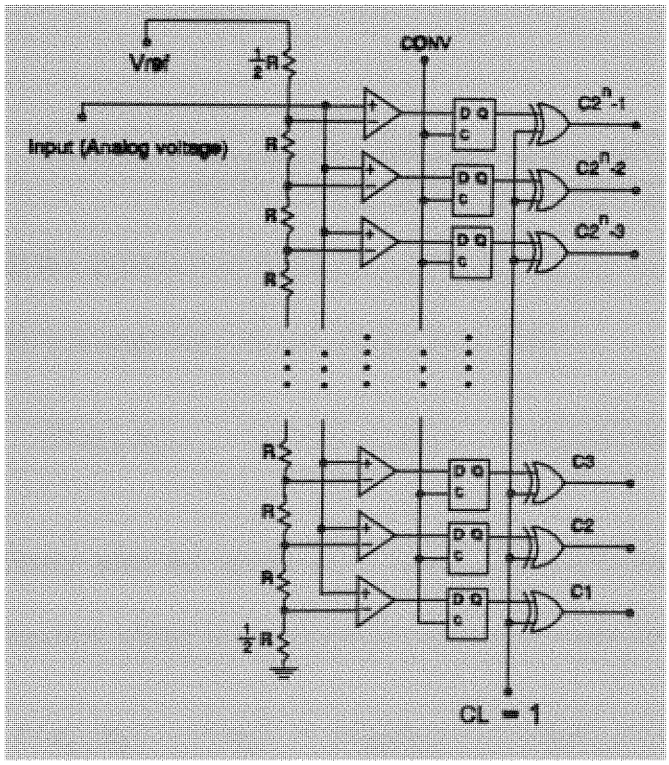


Figure 5.5. A n-bit flash analog to digital converter which is designed by using a set of monotone decreasing functions

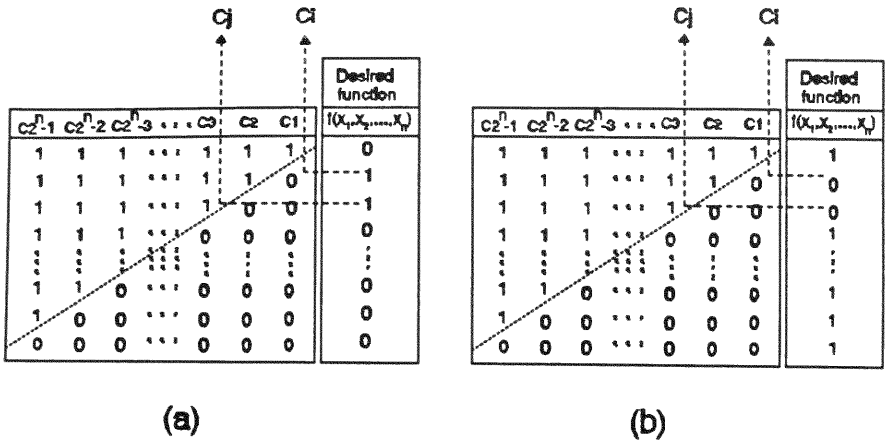


Table 5.10. (a) Function synthesis using logic one (minterm)
(b) Function synthesis using logic zero (maxterm)

After the operation of Table 5.10a, we can get the Boolean expression of a logic function which is expressed in sum of products terms.

$$f(x_1, x_2, \ldots, x_n) = C_{jn}C'_{in} + \ldots + C_{j2}C'_{i2} + C_{j1}C'_{i1} \qquad [5.4]$$

Similarly, after the operation of Table 5.10b, we can get the Boolean expression of a logic function which is expressed in product of sums terms.

$$f(x_1, x_2, \ldots, x_n) = (C'_{jn}C_{in}) \ldots (C'_{j2}C_{i2})\,(C'_{j1}C_{i1}) \qquad [5.5]$$

It was obviously that we can synthesize any function by using either Table 5.10a or Table 5.10b. As follows, I illustrate two examples to explain how these two methods really work.

Example 5.7. Minimum synthesis of three variables function $f(x_1,x_2,x_3)=\Sigma(1,2,6)$ by using a 3-bit monotone decreasing flash analog to digital converter in Table 5.11.

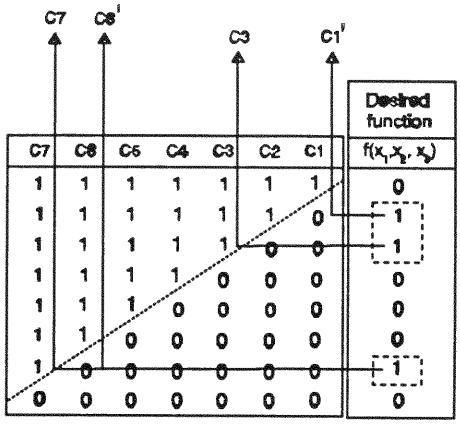


Table 5.11. Minimum synthesis of $f(x_1,x_2,x_3)=\Sigma(1,2,6)$ by using a 3-bit monotone decreasing flash A/D converter

In this example, since the # of segment of 0 which is equal to three is larger than the # of segment of 1 which is equal to two and both the first and the last bits are logic zeros, we apply Table 5.10a and equation 5.4. Then we can create the truth table 5.12.

C_7	C_6'	C_7C_6'	C_3	C_1'	C_3C_1'	$f(x_1,x_2,x_3) = C_7C_6'+C_3C_1'$
1	0	0	1	0	0	0
1	0	0	1	1	1	1
1	0	0	1	1	1	1
1	0	0	0	1	0	0
1	0	0	0	1	0	0
1	0	0	0	1	0	0
1	1	1	0	1	0	1
0	1	0	0	1	0	0

Table 5.12

As a result, the minimum function $f(x_1,x_2,x_3) = C_7C_6'+C_3C_1'$.

Example 5.8. Minimum synthesis of three variables function $f(x_1,x_2,x_3)=\Sigma(0,4,5,7)$ by using a 3-bit monotone decreasing flash analog to digital converter in Table 5.13.

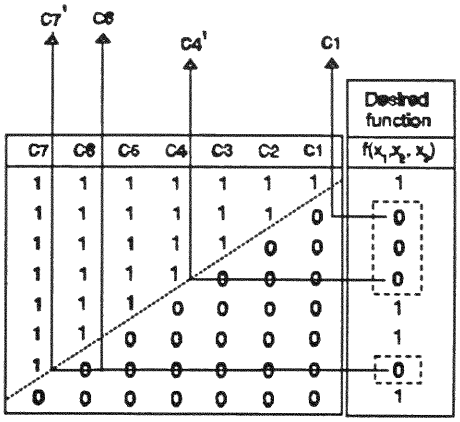


Table 5.13. Minimum synthesis of $f(x_1,x_2,x_3)=\Sigma(0,4,5,7)$ by using a 3-bit monotone decreasing flash A/D converter

In this example, since the # of segment of 0 which is equal to two is less than the # of segment of 1 which is equal to three and both the first and the last bits are logic ones, we apply Table 5.10b and equation 5.5. Then we can create the truth table 5.14.

C_7' C_6	$C_7'+C_6$	C_4' C_1	$C_4'+C_1$	$f(x_1,x_2,x_3) = (C_7'+C_6)(C_4'+C_1)$
0 1	1	0 1	1	1
0 1	1	0 0	0	0
0 1	1	0 0	0	0
0 1	1	0 0	0	0
0 1	1	1 0	1	1
0 1	1	1 0	1	1
0 0	0	1 0	1	0
1 0	1	1 0	1	1

Table 5.14

As a result, the minimum function $f(x_1,x_2,x_3) = (C_7'+C_6)(C_4'+C_1)$.

5.4 Conclusion

After the above illustrations, we can synthesize any given function by using either monotone increasing flash A/D converters or monotone decreasing flash A/D converters. The rules of how to synthesize a given function are also presented and illustrated by examples. It can be applied usefully to logic design, computer design, and other related areas.

Chapter Six

Fuzzy Classification

6.1 Introduction

After the discussion of fuzzy threshold functions, fuzzy monotone functions, and fuzzy unate functions, there are many results that can be generalized and we can find some relationships from these results. Furthermore, if a function is given, I try to classify this function that should be belonged to which special logic function.

The grade of membership function μ is normalized in the interval [0 1] and it is set to be 1 if the function is a specific special logic function. So general speaking, when we have many special logic functions and want to classify them, the larger grade of membership function μ, the stronger connection to the class function. Therefore, the classification is based upon fuzzy sets theory and I restrict that there are only four fuzzy special logic functions when I investigate the classification procedure, they are fuzzy threshold functions, fuzzy monotone increasing functions, fuzzy monotone decreasing functions, and fuzzy unate functions. After the discussion, readers will find that the classification procedure can be usefully applied to any other special logic functions.

6.2 Generalization

First of all, I summarize some important results of fuzzy threshold functions, fuzzy monotone functions, and fuzzy unate functions which I have already discussed in chapter 1, 2 and 3. According to these results, the relationships among them can be found and a procedure for fuzzy classification can be generalized.

6.2.1 Fuzzy threshold functions

No. of variables	No. of most inseparable functions	C_{max}	$\rho_n=1/C_{max}$	membership function
2	2	1	1	$\mu_T=1-C$
3	2	2	1/2	$\mu_T=1-(C/2)$
4	2	7	1/7	$\mu_T=1-(C/7)$
5	2	15	1/15	$\mu_T=1-(C/15)$
6	2	31	1/31	$\mu_T=1-(C/31)$
. . .	. . .	. . .	. . .	. . .
n	2	$2^{n-1}-1$	$1/(2^{n-1}-1)$	$\mu_T=1-C/(2^{n-1}-1)$

Table 6.1. Summary of fuzzy threshold functions

THEOREM 6.1. For all member of variables, the number of most inseparable functions are always equal to 2. They are odd function and even function.

THEOREM 6.2. f is a threshold function if and only if f' is a threshold function. So it can be generalized to fuzzy threshold function. For all function of n variables, the minimum number of minterms change of function f is equal to the minimum number of minterms change of complemented function f'. And so as to the grade of membership function μ_T.

$$C(f) = C(f') \qquad [6.1]$$

$$\mu_T(f) = \mu_T(f') \qquad [6.2]$$

6.2.2 Fuzzy monotone functions

No. of variables	No. of most variation functions	C_{max}	$\rho_n=1/C_{max}$	membership function
2	2	2	2	$\mu_M=1-(C/2)$
3	2	4	1/4	$\mu_M=1-(C/4)$
4	2	8	1/8	$\mu_M=1-(C/8)$
5	2	16	1/16	$\mu_M=1-(C/16)$
6	2	32	1/32	$\mu_M=1-(C/32)$
. . .	. . .	. . .	. . .	. . .
n	2	2^{n-1}	$1/2^{n-1}$	$\mu_M=1-C/2^{n-1}$

Table 6.2. Summary of fuzzy monotone functions

THEOREM 6.3. For all member of variables, the number of most variation functions are always equal to 2.

THEOREM 6.4. For all function f of n variables, if all the minterms are logic zeros or logic ones, then the function f is a monotone increasing function as well as a monotone decreasing function.

THEOREM 6.5. For all function of n variables, f is a monotone increasing function if and only if f' is a monotone decreasing function.

6.2.3 Fuzzy unate functions

No. of variables	No. of most non-unate functions	C_{max}	$\rho_n=1/C_{max}$	membership function
2	2	1	1	$\mu_U=1-C$
3	2	2	1/2	$\mu_U=1-(C/2)$
4	2	7	1/7	$\mu_U=1-(C/7)$
5	2	15	1/15	$\mu_U=1-(C/15)$
6	2	31	1/31	$\mu_U=1-(C/31)$
. . .	. . .	. . .	. . .	. . .
n	2	$2^{n-1}-1$	$1/(2^{n-1}-1)$	$\mu_U=1-C/(2^{n-1}-1)$

Table 6.3. Summary of fuzzy unate functions

THEOREM 6.6. For all member of variables, the number of most non-unate functions are always equal to 2. They are odd function and even

function.

THEOREM 6.7. f is a unate function if and only if f' is a unate function. So for all functions of n variables, the minimum number of minterms change of function f is equal to the minimum number of minterms change of complemented function f'. And so as to the grade of membership function μ_U.

$$C(f) = C(f') \qquad [6.3]$$

$$\mu_U(f) = \mu_U(f') \qquad [6.4]$$

From the summaries of fuzzy special logic functions, I conclude the following theorems.

THEOREM 6.8. The number of most inseparable functions, most variation functions, and most non-unate functions are all equal to 2. In particular, the two most inseparable functions are as same as the two most non-unate functions. They are all odd function and even function.

Since that, the algorithms of fuzzy threshold functions and fuzzy unate functions for generating these two functions are the same.

THEOREM 6.9. For any function of n variables, the maximum number of minterms change C of fuzzy threshold functions is the same as the value of fuzzy unate functions.

6.3 Relationships

After the generalization, we can find some relationships among threshold functions, monotone increasing functions, monotone decreasing functions, and unate functions.

THEOREM 6.10. [21] All threshold functions are unate functions. Therefore, all non-unate functions are inseparable functions. But, unate functions are not necessarily threshold functions.

This theorem can be drawn as Figure 6.1.

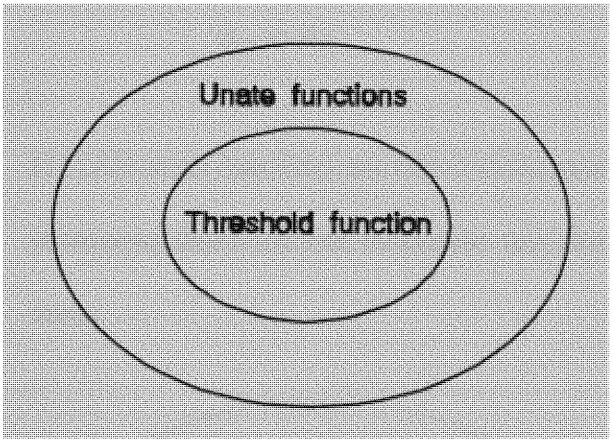


Figure 6.1. The relationship between threshold functions and unate functions

Example 6.1. $f_1(x_1,x_2,x_3) = x_1+x_2x_3$ and $f_2(x_1,x_2,x_3) = x_1'x_2'+x_3'$

are unate functions but not threshold functions.

Example 6.2. $f_3(x_1,x_2,x_3) = x_1x_2+x_1'x_2'$ and $f_4(x_1,x_2,x_3) = x_1x_2'+x_2x_3$

They are not unate functions and therefore they are not threshold functions.

Example 6.3. $f_5(x_1,x_2,x_3,x_4) = x_1x_2+x_3x_4$

is a unate function but not a threshold function.

From theorem 6.10, it can be applied to fuzzy threshold functions and fuzzy unate functions and get theorem 6.11.

THEOREM 6.11. For all functions f of n variables, the grade of membership function μ_U of a fuzzy unate function is always larger than the grade of membership function μ_T of a fuzzy threshold function.

$$\mu_U \geq \mu_T \qquad [6.5]$$

THEOREM 6.12. [15] All unate functions are mixed monotone functions -- monotone increasing functions and monotone decreasing functions, and vice versa.

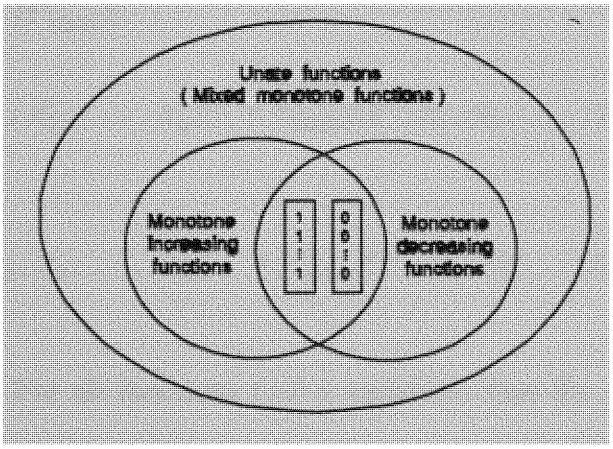


Figure 6.2. The relationship between unate functions and monotone functions

Theorem 6.12 can be applied to fuzzy unate functions and fuzzy monotone functions and we can get theorem 6.13.

THEOREM 6.13. For all functions f of n variables, the grade of membership function μ_U of a fuzzy unate function is always larger than the grade of membership function μ_M of a fuzzy monotone function.

$$\mu_U(f) \geq \mu_{MINC}(f) \qquad [6.6]$$

$$\mu_U(f) \geq \mu_{MDEC}(f) \qquad [6.7]$$

THEOREM 6.14. For all functions f of n variables, then the minimum number of minterms change in order to convert f to be a monotone increasing function (f_{MINC}) is equal to the minimum number of minterms change in order to convert f' to be a monotone decreasing function (f'_{MDEC}).

$$C_{MINC}(f(x_1, x_2, \ldots, x_n)) = C_{MDEC}(f'(x_1, x_2, \ldots, x_n)) \qquad [6.8]$$

$$C_{MDEC}(f(x_1, x_2, \ldots, x_n)) = C_{MINC}(f'(x_1, x_2, \ldots, x_n)) \qquad [6.9]$$

THEOREM 6.15. For all functions f of n variables, then the grade of membership function μ_{MINC} of a fuzzy monotone increasing function f is equal to the grade of membership function μ_{MDEC} of a fuzzy monotone decreasing function f'.

$$\mu_{MINC}(f(x_1, x_2, \ldots, x_n)) = \mu_{MDEC}(f'(x_1, x_2, \ldots, x_n)) \quad [6.10]$$

$$\mu_{MDEC}(f(x_1, x_2, \ldots, x_n)) = \mu_{MINC}(f'(x_1, x_2, \ldots, x_n)) \quad [6.11]$$

6.4 Classification

After the discussion of relationships of fuzzy threshold function, fuzzy monotone functions, and fuzzy unate functions, I wrote down a classification procedure in order to decide a function should be classified to which class function. Although there are many special logic functions, I restricted that I only classify four special function: fuzzy threshold functions, fuzzy monotone increasing functions, fuzzy decreasing functions, and fuzzy monotone functions in the topic.

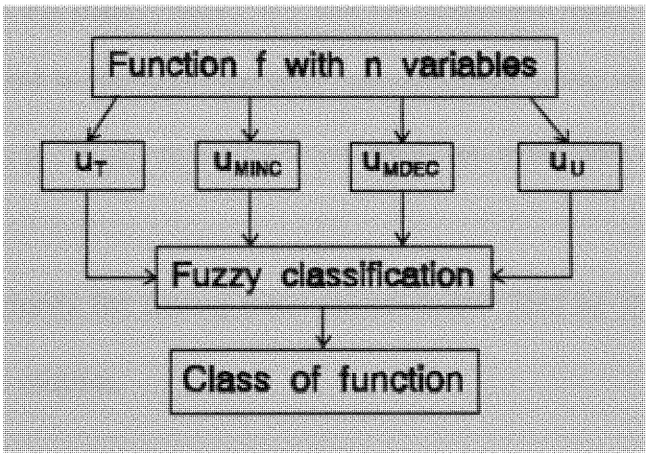


Figure 6.3. The concept of fuzzy classification

When a function f is being classified, I set a classification vector (CV) which includes all the grade of membership functions and set a threshold θ by user to decide whether this function should be classified or not.

$$CV = \max\{\mu_T, \mu_U, \mu_M\}_\theta \qquad [6.12]$$

A procedure for classifying fuzzy special logic functions

Step 1. A threshold θ is provided by the user where $0 \le \theta \le 1$.

Step 2. For a function f, the grade of membership functions μ_T, μ_U, μ_{MINC}, and μ_{MDEC} are computed.

Step 3. IF $\max\{\mu_T, \mu_U, \mu_{MINC}, \mu_{MDEC}\} < \theta$, THEN

f is not being classified.

Step 4. IF $\max\{\mu_T, \mu_U, \mu_{MINC}, \mu_{MDEC}\} \ge \theta$ and the maximum value is unique, THEN

f is classified to the class function corresponding to the maximum value.

Step 5. IF $\max\{\mu_T, \mu_U, \mu_{MINC}, \mu_{MDEC}\} \ge \theta$ and there are more than one maximum value, THEN

Either a priority list among these classes is provided by the user or all maxima classes are outputs.

Example 6.4. For a three-variable function $f_2(x_1,x_2,x_3) = x_1+x_2'x_3$

Step 1. To set a threshold θ which equals to 0.5. If $\max\{\mu_T, \mu_U, \mu_{MINC}, \mu_{MDEC}\} < 0.5$, we decide not to classify this function. But if $\max\{\mu_T, \mu_U, \mu_{MINC}, \mu_{MDEC}\} \geq 0.5$, we decide to classify this function.

Step 2. To calculate the grade of membership functions μ_T, μ_U, μ_{MINC}, and μ_{MDEC}.

It is a threshold function and the grade of membership function μ_T is equal to 1. It is also a unate function and the grade of membership function μ_U is equal to 1.

But it is neither a monotone increasing function nor monotone decreasing function. If we want to convert $f_2(x_1,x_2,x_3)=\Sigma(1,3,4,5,7)$ into a monotone increasing function $f_{MINC}(x_1,x_2,x_3)$, then it becomes as $f_{MINC}(x_1,x_2,x_3)=\Sigma(3,4,5,6,7)$ and the number of minterms changes $C(f(x_1,x_2,x_3))=2$. So $\mu_{MINC}=(2/4)=0.5$. If we want to convert $f_2(x_1,x_2,x_3)=\Sigma(1,3,4,5,7)$ into a monotone decreasing function $f_{MDEC}(x_1,x_2,x_3)$, then it becomes as $f_{MDEC}(x_1,x_2,x_3)=\Sigma(0,1,2,3,4,5)$ and $C(f(x_1,x_2,x_3))=3$. So the grade of membership function $\mu_{DEC}=(3/4)=0.75$.

Step 3. To decide the function should be classified to which class function.

Now $\max\{\mu_T, \mu_U, \mu_{MINC}, \mu_{MDEC}\} = \max\{1,1,0.5,0.75\} = 1 \geq 0.5$. According to step 5 of the algorithm, we have the following two possible results.

(1) All the maxima classes are outputs. So this function is a fuzzy threshold function and it also is a fuzzy unate function.

(2) According to theorem 6.11, we can say that a fuzzy threshold function is superior to a fuzzy unate function. As a result, we classify this function as a fuzzy threshold function.

6.5 Conclusion

Fuzzy threshold functions, fuzzy monotone functions, and fuzzy monotone functions are discussed and summarized. Using fuzzy logic theory to classify a function is discussed and a procedure for classifying fuzzy special logic functions is also presented. Although I only discuss four fuzzy special logic functions here, this procedure is useful to extend to all fuzzy special logic functions. And since the threshold θ is decided by user, the classification becomes more flexible.

Chapter Seven

Recommendations

(1) It is essential to apply fuzzy logic to the other special logic functions.

(2) It is necessary to investigate these four fuzzy special logic functions with don't-care conditions.

(3) It is worthwhile to identify each group of fuzzy special logic function by its distance (the minimum minterms change) to full membership functions.

(4) Since all the logic functions can be represented by fuzzy special logic functions, the results of these four fuzzy special logic functions may be applied to pattern recognition.

References

(1) C. L. Phillips and H. T. Nagle, Digital Control System Analysis and Design, 1990.

(2) Daniel H. Sheingold, Analog-Digital Conversion Handbook, 1986.

(3) E. A. Parr, The Logic Designer's Guidebook, 1984.

(4) E. T. Lee, "Shape-Oriented Chromosome Classification", IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-5, No.6, November, 1975, pp.629-632.

(5) E. T. Lee and L. A. Zadeh, "Note on Fuzzy Languages", Information Sciences, Vol.1, 1969, pp.421-434.

(6) E. T. Lee, "Pictorial Data Compression Using Array Grammars", Proc. of the Data Compression Conference, April 8-10, 1991, Snowbird, Utah.

(7) E. T. Lee, "Pictorial Data Compression Techniques", Proc. of the Data Compression Conference, April 8-10, 1991, Snowbird, Utah.

(8) Eugene R. Hnatek, A User's Handbook of D/A and A/D Converters, 1976.

(9) H. T. Nagle, Jr., B. D. Carroll, and J. D. Irwin, An Introduction to Computer Logic, 1975.

(10) James C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, 1981.

(11) John D. Lenk, Logic Designer's Manual, 1977.

(12) Johnson, Everett L., Digital Design, 1987.

(13) L. A. Zadeh, Fuzzy algorithms, 1968.

(14) M. Morris Mano, Digital Design, 1991.

(15) Michael A. Harrison, Introduction to Switching and Automata Theory, 1965.

(16) P. M. Lewis II and C. L. Coates, Threshold Logic, 1967.

(17) P. P. Wang and S. K. Chang, Fuzzy Sets, 1980.

(18) Sankar K. Pal and Dwijesh K. Dutta Majumder, Fuzzy Mathematical Approach to Pattern Recognition, 1986.

(19) S. C. Lee and E. T. Lee, "Fuzzy Neural Networks", Mathematical Biosciences, Vol.23, 1975, pp.151-177.

(20) Thomas C. Bartee, Digital Computer Fundamentals, 6th Edition, 1985.

(21) Zvi Kohavi, Switching and Finite Automata Theory, 1970.