NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

VLSI IMPLEMENTATION OF
FUZZY LOGIC
OPERATOR UNIT

by

Ismail bin Dewa

June, 1991

Thesis Advisor: Chyan Yang

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL *(If applicable)* 33	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000		7b. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS *(City, State, and ZIP Code)*		10. SOURCE OF FUNDING NUMBERS	
		Program Element No	Project No / Task No / Work Unit Accession Number

11. TITLE *(Include Security Classification)*
VLSI IMPLEMENTATION OF FUZZY LOGIC OPERATOR UNIT

12. PERSONAL AUTHOR(S) Ismail bin Dewa

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(year, month, day)*	15. PAGE COUNT
Master's Thesis	From To	June 1991	56

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS *(continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUBGROUP	Fuzzy logic, max, min, inv

19. ABSTRACT *(continue on reverse if necessary and identify by block number)*

Fuzzy logic is widely used in many applications that deal with uncertainty and approximate reasoning in decision making. Decisions can be made based on fuzzy inferences. Because of the ease with which Very Large Scale Integration (VLSI) circuits can be made, hardware implementation of fuzzy logic is thus seen to be an appropriate step to be taken to fully realize its potential. Fuzzy operations are based on three basic operators, the maximum, minimum and inverse functions. This thesis investigates its implementation in VLSI circuits, specifically for digital systems. Design structures such as bit-cascade, bit-slice, block-cascade and block-slice were implemented. Comparisons between these designs are provided.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS		
22a. NAME OF RESPONSIBLE INDIVIDUAL Chyan Yang	22b. TELEPHONE *(Include Area code)* (408) 646-2266	22c. OFFICE SYMBOL EC/Ya

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for public release; distribution is unlimited.

VLSI Implementation of
Fuzzy Logic
Operator Unit

by

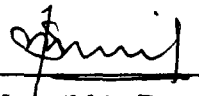
Ismail bin Dewa
Lieutenant Commander, Royal Malaysian Navy
B.S., Teesside Polytechnic, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1991

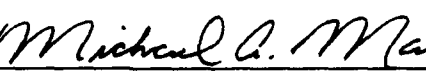
Author: 

Ismail bin Dewa

Approved by: 

Chyan Yang, Thesis Advisor

Jon T. Butler, Co-Advisor



Michael A. Morgan, Chairman
Department of
Electrical and Computer Engineering

ABSTRACT

Fuzzy logic is widely used in many applications that deal with uncertainty and approximate reasoning in decision making. Decisions can be made based on fuzzy inferences. Because of the ease with which Very Large Scale Integration (VLSI) circuits can be made, hardware implementation of fuzzy logic is thus seen to be an appropriate step to be taken to fully realize its potential. Fuzzy operations are based on three basic operators, the maximum, minimum and inverse functions. This thesis investigates its implementation in VLSI circuits, specifically for digital systems. Design structures such as bit-cascade, bit-slice, block-cascade and block-slice were implemented. Comparisons between these designs are provided.

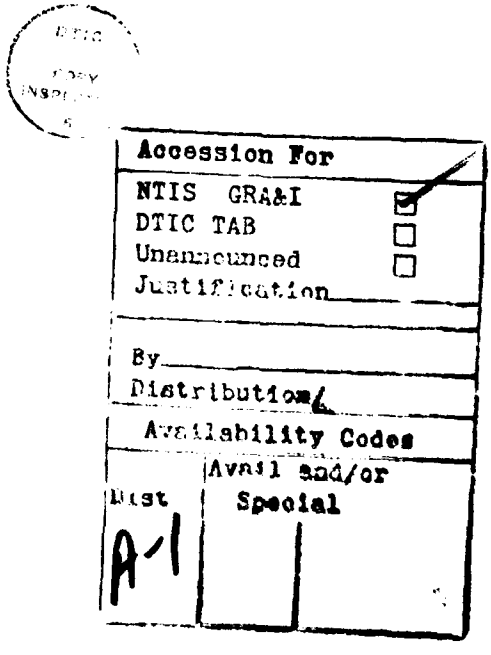


TABLE OF CONTENTS

I. INTRODUCTION 1

II. FUZZY LOGIC 3

A. FUZZY SET 4

B. FUZZY SYSTEMS 6

III. FUZZY LOGIC OPERATOR CIRCUIT DESIGN 12

A. NUMBER SYSTEM 12

B. INV CIRCUIT DESIGN 13

C. MAXIMUM AND MINIMUM OPERATOR UNIT DESIGN . . . 14

1. Serial Design 15

2. Parallel Design 17

a. Cascade MIN Design 18

b. Bit-Slice Design 21

c. Block Cascade/Slice Design 23

IV. VLSI IMPLEMENTATION PROCEDURES 26

A. CMOS TRANSISTOR CIRCUIT 27

B. VLSI CAD TOOLS 29

1. Magic 29

2. RNL 30

3. SPICE 30

C. VLSI CIRCUIT LAYOUT OF FUZZY OPERATOR UNIT . . 31
D. LAYOUT VERIFICATION 32
E. TIMING ANALYSIS 33
F. RESULTS 34
1. VLSI Estate Area 34
2. Transistor Count 35
3. Timing Analysis 36

V. CONCLUSIONS 39

APPENDIX 41

LIST OF REFERENCES 46

INITIAL DISTRIBUTION LIST 49

I. INTRODUCTION

Fuzzy set theory was first introduced by Zadeh [Ref. 1] as a generalization of ordinary set theory. This generalization has led to applications such as pattern classification, information processing, control, artificial intelligence and, more generally, decision processes involving incomplete and uncertain data.

Fuzzy logic, which is based on fuzzy set theory, is sometimes defined as the effective methodology for treating imprecise human knowledge. It tries to deal with the uncertainty and imprecision inherent in human decision making and reasoning processes. At present, especially in Japan, the most direct and immediate application for fuzzy logic has been the development of control devices. [Ref. 2,3]

Human decision and reasoning is usually based on fuzzy inferences that use past knowledge. To emulate this approximate reasoning process, expert systems used past human knowledge to form its IF-THEN rules. Often LISP and PROLOG are used to implement these rules. Thus, fuzzy inferences are implemented through software using existing Boolean logic hardware. The hardware realization of fuzzy logic can thus be seen as the appropriate step to fully realize a fuzzy inference application. This objective was clearly stated by Han and Singh [Ref. 4]. This thesis is an extension of the

paper by Han and Singh and focuses on the VLSI design and implementation of fast fuzzy logic operators in digital systems. The basic fuzzy logic operations are the maximum, minimum and inversion functions. Each of these operations was implemented and investigated through the use of VLSI CAD tools available at the US Naval Postgraduate School namely MAGIC, RNL, PRESIM and SPICE2 [Ref. 5,6,7]. Various circuit designs of the operators were developed and comparisons were made on their VLSI estate size, transistor count and speed performance. Based on these comparisons, the circuit giving the best performance could be used as building blocks for constructing real-time fuzzy controllers and inference engines.

The organization of this thesis is described as follows. Chapter II gives an overview of fuzzy logic and concepts pertinent to the thesis. Chapter III discusses the circuit design of the various Fuzzy Operation Units(FOU) [Ref. 4]. Chapter IV deals with the procedures of implementing these designs as well as the experimental results. Chapter V concludes the thesis.

II. FUZZY LOGIC

This chapter gives a condensed overview of fuzzy logic from various references [Ref. 8,9,10,11,12,13,14].

Fuzzy logic may be viewed as an extension of multivalued logic(MVL) [Ref. 8]. However, its uses and objectives are quite different from the traditional MVL. Fuzzy logic deals with approximate rather than precise modes of reasoning. It treats everything, including truth, as a matter of degree.

Boolean logic comes about partly due to the relative simplicity of designing binary switching systems and partly because most of the basic switching modules are two-valued or bistable. Thus, it is sometimes not adequate for handling real world problems. In fuzzy logic the truth value of a formula, instead of assuming one of the two values (0 and 1), can assume any value in the interval of [0,1] and is used to indicate the degree of truth represented by the formula. For example, if the proposition P(x) represent 'x is large compared with unity' then the truth value of $P(10^6)$ and $P(10^{-6})$ are certainly 1 and 0 respectively. As for P(125), the truth value of it may be some value between 0 and 1, say 0.25. This truth value is called the membership grade or membership function and is denoted as $\mu_p(125) = 0.25$.

Some of the main features that differentiate fuzzy logic from the traditional logical systems are stated by Zadeh [Ref. 6] as follows:

- In two-valued logical systems, a proposition is either true or false. In multivalued logic, it may be true or false or have an intermediate truth value. In fuzzy logic, the truth values are allowed to range over the interval of 0 and 1.
- Two-valued and multivalued logics allow only the quantifiers "all" and "some". Fuzzy logic allows fuzzy quantifiers such as "most", "many", "several", "few", "frequently", "occasionally", "about ten" and so on.

A. FUZZY SET

A fuzzy set is a set and a function (membership function) f mapping elements of the set into the interval [0,1]. If x is an element of a fuzzy set, then $f(x)$ is called the grade of x.

Mathematically, if the symbol X denotes a universe of discourse, which may be an arbitrary collection of subjects or mathematical constructs, then A, which is defined as a finite subset of X, can be expressed as

$$A = \{x_{i_1}, x_{i_2}, \ldots, x_{i_m}\},$$

and

$$i_j \in \{1, 2, \ldots, m\}$$

where $x_{i_j} \in \{x_1, x_2, \ldots\ldots, x_n\}$ is an element X and n is the number of elements in X.

A finite fuzzy subset of A on X is then a set of ordered pairs $\{(x_i, \mu_A(x_i))\}$ where $\mu_A(x_i)$ represent grade of membership or degree of membership. If $\mu_A(x_i)$ takes only the value of 0 or

1 then $\mu_A(x_i)$ is in effect the usual binary Boolean function. However, $\mu_A(x_i)$ usually lies in the interval [0,1] with 0 and 1 denoting non-membership and full-membership respectively. It should also be pointed out that fuzzy subset A can also be denoted by just its membership function, $\mu_A(x_i)$. These two denotions are used interchangeably in many papers on fuzzy logic.

An example to illustrate a fuzzy set is as follows. Referring to Figure 1, let the desirable properties of selecting a husband by a woman be the universe of discourse such that

$$X = \{ \text{young, handsome, rich} \}$$

and let D be the range set the membership function $f(x)$ maps into. For this example, we have chosen the range of $f(x)$ to be discretized levels. In reality the range can be continuous.

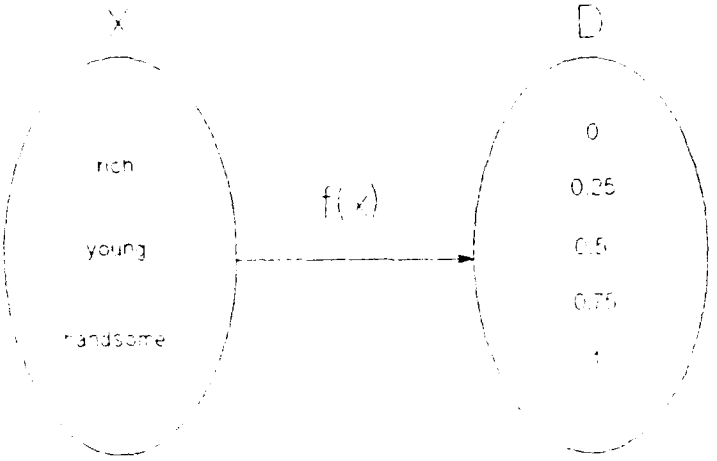


Figure 1 Fuzzy Set

Thus the fuzzy subset A is

$$A = \{(x_i, \mu_A(x_i)) \mid \mu_A(x_i) \in D, \quad x_i \in X\}$$

where x_i can be the element young, rich, handsome

and

$\mu_A(x_i)$ can take any one of {0, 0.25, 0.5, 0.75, 1}

The subset A can be one of the 125 possible fuzzy subsets, since each membership function of element X has five possibilities. The following are examples of subset A,

{(young,0.5)}

or {(rich,0.75)}

or {(young,0.25),(rich,1)} and so on.

Note that sometimes we indicate (young,0.5) as $\mu_{young}(x) = 0.5$ for an element x in X.

If D has only 0 and 1 as its element then A is the regular Boolean subset that is for example if

$$A = \{(young, \mu_A(young)\}$$

then if $\mu_A(young) = 1$, young is in the set A

$= 0$, young is not in the set A

B. FUZZY SYSTEMS

In Boolean logic, the basic logic operations are AND, OR and NOT. The rest of the logic operations such as NAND, NOR, EXCLUSIVE OR are derived from these basic operations. The same can be said about fuzzy logic operations. Their basic

operations are the maximum(MAX), minimum(MIN) and inversion(INV). Rules for fuzzy inference system can usually be formed from these operations.

As explained previously, a fuzzy subset A of a universe of discourse X is characterised by a membership function $\mu_A(x)$. The three basic operators used in a fuzzy set are defined as follows:

1. The union of the fuzzy subsets A and B of the universe of discourse X is a fuzzy subset denoted by $A \cup B$, with a membership function defined by
$$\mu_{A \cup B}(x) = \max[\mu_A(x); \mu_B(x)]$$
The union corresponds to the Boolean connective 'OR'.

2. The intersection of the fuzzy subsets A and B is a fuzzy subset, denoted $A \cap B$, with a membership function defined by
$$\mu_{A \cap B}(x) = \min[\mu_A(x); \mu_B(x)]$$
Theintersection corresponds to the connective 'AND'.

3. The inversion(complement) of a fuzzy subset is a fuzzy subset denoted by ¬A, with a membership function defined by
$$\mu_{\neg A}(x) = 1 - \mu_A(x)$$
Complementation corresponds to the 'NOT' operator.

An example describing these operations are as follows

Using the example from the previous section, let

$$A = \{ (\text{young}, 0.5) \}$$

and

$$B = \{ (\text{rich}, 0.75) \}$$

and

$$C = \{ (\text{handsome}, 0.25) \}$$

then if S is the fuzzy subset such that properties selected was 'handsome or rich and not young' then S can be stated as

$$S = (\ C\ OR\ B\ )\ AND\ (\ NOT\ A\ ).$$

The numerical truth value of S is

$$\begin{aligned}\mu_S(x) &= \min\{\max(\mu_C,\mu_B),1-\mu_A\}\\ &= \min\{\max(0.25,0.75),1-0.5\}\\ &= \min\{0.75,0.5\}\\ &= 0.5.\end{aligned}$$

The definition of a fuzzy set helps us to deal with information pertaining to human experience. As an example, consider a mixing process in a chemical plant. A human operator will use his past knowledge to optimize the flow of a fluid against the temperature of the process. This operator knowledge is easily translated to a set of rules in a fuzzy system whereas translation in the traditional control system design might lead to a complex mathematical model which may be impractical to implement.

A basic concept in fuzzy logic is that of a linguistic variable or fuzzy variable. A linguistic variable is defined by Zadeh as a variable whose values is a word or sentence in a natural or synthetic language. For example, 'flow' is a linguistic variable since its values is contained in linguistic expressions such as the flow is 'big', 'medium', 'small', etc. Here, 'big', 'medium' and 'small' are labels to fuzzy sets. To describe a controlling action, fuzzy variables are used in fuzzy rules. An example of such a rule is the statement : if 'input is big' then 'output is medium'.

'Input' and 'output' are fuzzy variables describing the fuzzy rule.

A fuzzy rule is defined as the relation between observation and action or between two fuzzy variables. A fuzzy relation R from a set X to a set Y is a fuzzy subset of the cartesian product $X \times Y$. $X \times Y$ is the collection of ordered pairs (x,y): $x \in X$, $y \in Y$. The membership function of fuzzy relation, for example the relation R : if A then B; given the fuzzy subset A of the universe of discourse X and the fuzzy subset B of Y, is defined as

$$\mu_R(x,y) = \min[\mu_A(x);\mu_B(y)] \qquad x \in X, y \in Y$$

A simple example to illustrate this, is suppose that we have

$$X = \{ 1,2,3\}$$

and

$$Y = \{ 1',2',3',4' \}.$$

Then

$$R = X \times Y = \{ (1,1'),(1,2'),(1,3'),(1,4'),(2,1'),(2,2'),$$
$$(2,3'),(2,4'),(3,1'),(3,2'),(3,3'),(3,4') \}.$$

Let $\mu_A(x)$ and $\mu_B(y)$ be defined as

x	$\mu_A(x)$	y	$\mu_B(y)$
1	0.0	1'	0.1
2	0.5	2'	0.9
3	1.0	3'	0.9
		4'	0.1

Therefore $\mu_R(x,y)$ is given as

x	y	$\mu^R(x,y)$
1	$1'$	0.0
2	$2'$	0.0
3	$3'$	0.0
1	$4'$	0.0
2	$1'$	0.1
3	$2'$	0.5
1	$3'$	0.5
2	$4'$	0.1
3	$1'$	0.1
1	$2'$	0.9
2	$3'$	0.9
3	$4'$	0.1

A linguistic interpretation can be given to this example. That is, $\mu_A(x)$ can be viewed as "input is big" and $\mu_B(y)$ can be viewed as "output is medium", where x and y represent input and output respectively. Then $\mu_R(x,y)$ can be viewed as "if 'input is big' then 'output is medium'". For example $\mu_R(x,y)$ achieves its maximum value, when x is 3 (input is indeed big) and y is $2'$ or $3'$ (output is indeed medium).

A controller system can be described by a set of such fuzzy rules for example

if 'input is big' then 'output is medium'

or(else)

if 'input is medium' then 'output is small'

This can be translated to an overall fuzzy relation R of two fuzzy implications through the use of the connective 'OR'. Thus, R of (if A_1 then B_1 or(else) if A_2 then B_2) has the membership function

$$\mu_R(y,x) = \max[\min[\mu_{A_1}(x);\mu_{B_1}(x)];\min[\mu_{A_2}(x);\mu_{B_2}(x)]]$$

This can be extended to cases of more than two fuzzy implications.

Suppose we have a fuzzy observation A' and the overall relation R. For our case A' can be an input between big and medium, what action should we take? The resultant action is inferred by the compositional rule of inference [Ref. 9], that is

$$B' = A' \circ R$$

The membership function of B' is defined by

$$\mu_{B'}(y) = \max_x \min(\mu_{A'}(x);\mu_R(y,x))$$

Another example of a fuzzy system is an inference engine. The engine will produce an output based on specific number of if-then-else rules.

From the above discussion we see the importance of the basic fuzzy operators (MAX, MIN and INV) in the implementation of a fuzzy system such as a fuzzy controller and an inference engine.

III. FUZZY LOGIC OPERATOR CIRCUIT DESIGN

From the previous chapter, we know that a fuzzy inference system can be implemented using the basic building blocks of MAX, MIN and INV. MAX and MIN are the most significant fuzzy logic operators since many fuzzy inference processors employ them in many applications. In this thesis, we will go through the design process of all the three operator units. These units basically operate on the numerical values of the membership function μ. Since these values lie in the interval of 0 and 1 we need to define a number system that we use in the thesis work.

A. NUMBER SYSTEM

Here, we employ a number system which is similar in representation to binary fractional numbers. We represent a fuzzy number as

$$N = .bbbbbb.....b_n$$

where n depends on the number of bits used to discretize the membership function of a fuzzy subset. The binary point here is implied. Each fuzzy number is $1/(2^n-1)$ less than or greater than the succeeding or preceding fuzzy number, respectively. Thus we see that bit weighting is not the same as a binary fractional number. To convert a fuzzy number to its decimal

equivalent, multiply its binary weight by $1/(2^n-1)$. Note that for n=4 the fuzzy number

$$0001 = 1 \times (1/2^4-1)$$

$$= 0.0667.$$

Table 1 shows a fuzzy number system with n=4 .

TABLE 1 FUZZY NUMBER REPRESENTATION

Fuzzy Number	Decimal Equivalent
0000	0.0000
0001	0.0667
0010	0.1333
0011	0.2000
0100	0.2667
0101	0.3333
0110	0.4000
0111	0.4667
1000	0.5333
1001	0.6000
1010	0.6667
1011	0.7333
1100	0.8000
1101	0.8667
1110	0.9333
1111	1.0000

B. INV CIRCUIT DESIGN

This is the simplest of the three designs. By definition,

$$INV(A) = 1 - \mu_A(v).$$

Accordingly, this will requires a substractor circuit. But by virtue of the fuzzy number system we choose we avoid this; instead only INVERTERS are required for our INV operator unit. As an example

INV (0010) = (1101).

A schematic diagram for INV is shown in Figure 2.

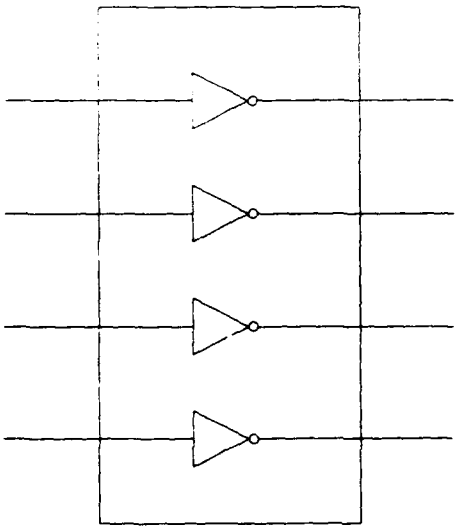


Figure 2 Inversion Circuit Diagram

C. MAXIMUM AND MINIMUM OPERATOR UNIT DESIGN

As implied by its meaning, the MIN operator takes two fuzzy numbers and produces the smaller of them, while the MAX operator produces the larger. Thus MAX/MIN is basically a magnitude comparator operating on two binary fuzzy numbers.

Two n-bit numbers can be compared with each other either serially or in parallel. Accordingly, the design of the MAX/MIN operator will follow this concept.

1. Serial Design

The basic principle in comparing two n-bit numbers is to compare their most significant bit(MSB) first. A 1 or 0 MSB will indicate that the number is greater or less than if the other number MSB is a 0 or 1 respectively. If the MSB is the same, the next significant bit is compared until an unequal bit is found. If all bits are the same the two numbers is equal.

In the serial design, the bits of the two numbers are compared in a serial fashion with the MSB first into the MAX/MIN operator unit. A state diagram for a serial MAX unit is shown in Figure 3.

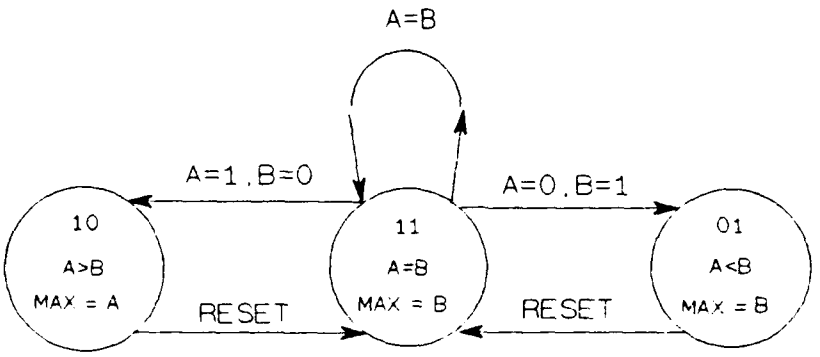


Figure 3 Bit Serial MAX State Diagram

In Figure 3, a reset input places the unit in the A=B state and as long as A and B bits are equal it remains in this state and the MAX output takes the value of B. When A nd B

bits differ, the MAX unit either goes to the A>B or A<B state until the next reset input. The MAX output takes the value of A in the A>B state and the B value in the A<B state. The MAX operation is expected to be slow using this serial design, since if there are n-bits per fuzzy number, the unit will takes at least (n + 1) clock cycles including a reset pulse to complete an operation. A possible circuit realization of this design is as shown in Figure 4.

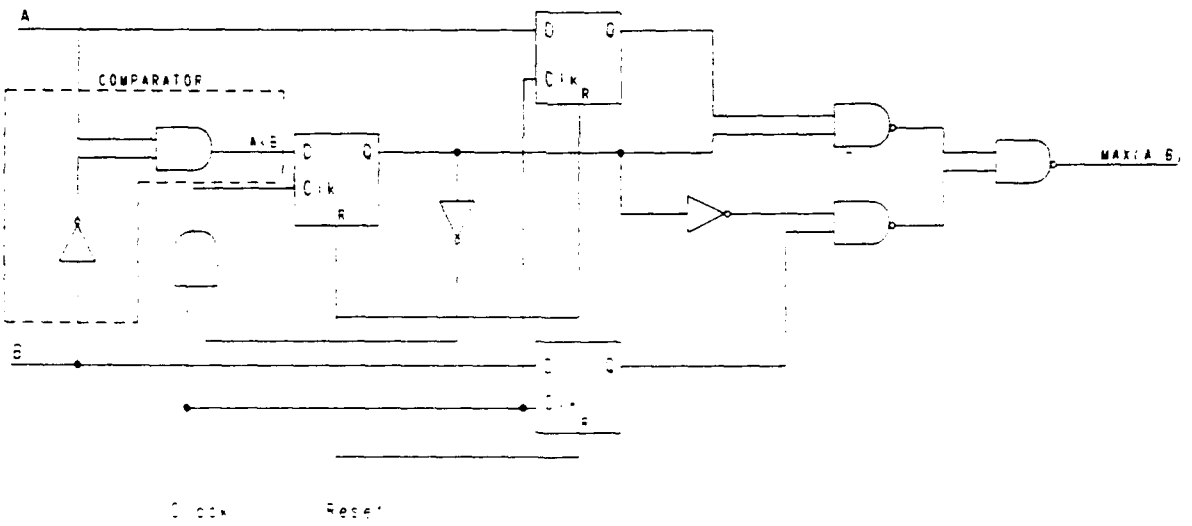


Figure 4 Bit Serial MAX Circuit Diagram

A serial MIN unit can similarly be designed. A MIN unit only needs the comparator circuit of Figure 4 be replaced by the circuit shown in Figure 5.

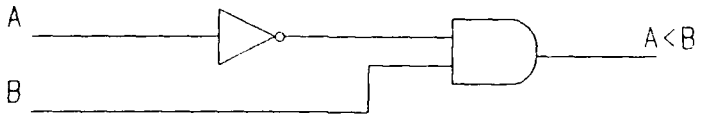


Figure 5 A<B Comparator Circuit

2. Parallel Design

In the serial design, the two n-bit fuzzy numbers, *A* and *B*, are compared a bit at a time with the MSB first. The fuzzy numbers are presented serially to the MAX/MIN unit. For the parallel design, the two numbers are compared n-bits at a time in parallel. As explained in the previous section, the same basic principle for magnitude comparison still applies here. Thus, although the n-bit numbers are compared in parallel, the comparison is still a bit by bit process in some sense. The preceding significant bit positions comparison decision either has to ripple through to the succeeding bit positions (the longest path is from the MSB to LSB) or be transmitted simultaneously to the succeeding bit positions (compare look-ahead). The scheme with the preceding comparison decision bit propagating to all succeeeding bit positions is the cascadable design, whereas the scheme using compare look-

ahead is defined in the thesis to be a bit-slice design. The above distinction between cascadable and bit-slice design is comparable to the well known adder circuit design (ripple-adder and carry look-ahead adder).

The cascadable design can be expected to be slower compared to the bit-slice design due to the propagation delay of the 'comparison-information' bit. A compromise design is one that breaks up the n-bit number into blocks of smaller bits, where each block is designed in a bit-slice structure. To form an n-bit MAX/MIN unit, these blocks can either be connected in a block-cascade or a block-slice structure. In the block-cascade structure, each block's output depends on the compare decision of the preceding blocks. This compare decision bit propagates from block to block. In the block-slice structure, each block is independent of previous or succeeding block compare decisions.

The following sections will give a detailed explanation on these design concepts. For simplicity, we will, however, be looking at a MIN operator unit design since a MAX can be designed by just replacing the comparator circuit as explained in the previous section.

a. Cascade MIN Design

Lets define the two fuzzy numbers to be compared as

$$A = (a_{n-1}, a_{n-2}, \ldots . a_i \ldots . a_0)$$

$$B = (b_{n-1}, b_{n-2}, \ldots . b_i \ldots . b_0)$$

where

a_i, b_i is the ith bit of A and B respectively

and the MIN output as

$$O = (o_{n-1}, o_{n-2}, \ldots . o_i \ldots . o_0)$$

where

o_i the ith bit of the output O.

for instance,

if $A = (0011)$

and $B = (1010)$

then $O = (0011)$.

Let's consider the i-th stage of the MIN operation. The output o_i will not only depend on the comparison of a_i and b_i but also on the (i+1)th stage comparison. Thus

if $a_{i+1} < b_{i+1}$ then $o_i = a_i$

if $a_{i+1} = b_{i+1}$ then $o_i = a_i$ if $a_i < b_i$

$= b_i$ if $a_i > b_i$

We can now define the propagating 'compare-decision' indicator bit from the (i+1)th stage as follows:

$c_{i+1} = 1$ if $a_{i+1} = b_{i+1}$

$g_{i+1} = 1$ if $a_{i+1} < b_{i+1}$

For the ith stage, we define the following two local indicators:

$z_i = 1$ if and only if $a_i = b_i$

$y_i = 1$ if and only if $a_i < b_i$

From these definitions, the following equations were derived [Ref. 4] :

$$g_i = g_{i+1} + c_{i+1}y_i \qquad (1)$$

$$c_i = c_{i+1}z_i \qquad (2)$$

$$o_i = a_i g_i + b_i g_i \qquad (3)$$

A block schematic of a 1 bit MIN unit is as shown in Figure 6 [Ref. 4] . Figure 7 shows the Boolean logic gates realization of the above equations. To form an n-bit MIN unit, n of this 1 bit MIN will be cascaded.

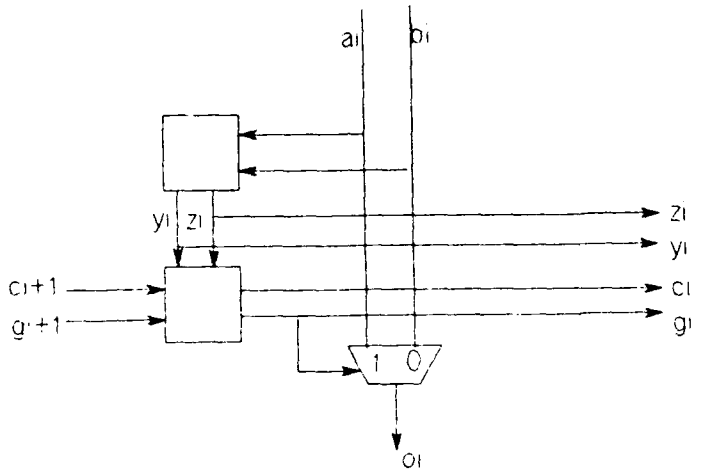


Figure 6 Block Schematic of 1 bit MIN

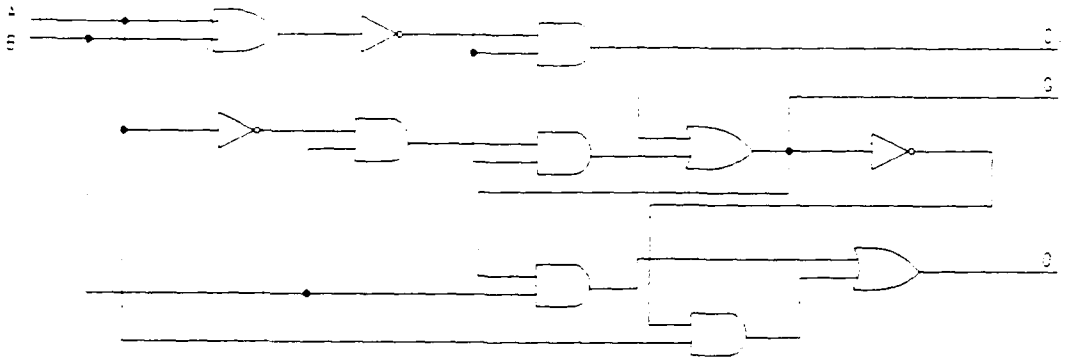


Figure 7 1 bit MIN Circuit Diagram

b. Bit-Slice Design

In the cascade design, a feature that slows down the MIN operation is the propagation delay caused by the ripple effect of the c_i and g_i variables from the MSB to the LSB. Bit-slice overcomes this by employing comparison look-ahead. Thus, each bit position of the output is independent of the decisions of the preceding stages.

Examining the recurrence equations (1) and (2), we see that it has a recursive property. Applying this property, we can generate c and g for each bit position independently. For example, c and g placed into each bit position of a 4-bit MIN unit are,

$$c_4 = 1$$

$$c_3 = c_4 z_3$$

$$c_2 = c_3 z_2 = c_4 z_3 z_2$$

$$c_1 = c_2 z_1 = c_4 z_3 z_2 z_1$$

$$g_4 = 0$$

$$g_3 = g_4 + c_4 y_3$$

$$g_2 = g_3 + c_3 y_2 = g_4 + c_4 y_3 + c_4 z_3 y_2$$

$$g_1 = g_2 + c_2 y_1 = g_4 + c_4 y_3 + c_4 z_3 y_2 + c_4 z_3 z_2 y_1$$

Observe that the variables c_4 and g_4 in the expressions for c_1, c_2, c_3, g_1, g_2 and g_3 are actually redundant since they are always connected to logic "1" and "0" respectively. Note that for an n-bit MIN unit, c_n is always set to logic "1" while g_n is always set to logic "0".

Other than the input a_i and b_i of equation 3 we see that the output is only affected by g. Notice that after applying the recursive properties of equations (1) and (2), the output does not require each bit position to generate the variable c. In the example given, compare-decision indicators g's are only dependent on c_4 and g_4 which are constants and the *AND* values of other local y and z indicators. For 4-bit MIN, the compare-decision bit placed in the LSB position, g_1, needs a 4 input *AND* for the last product term and a 4 input *OR* for the sum of all the product terms. Thus, as the number of bits increases, the hardware implementation of g_1 requires a higher

local indicators z_i and y_i from the higher significant bit cell to the lower significant bit cell will also increase. This is true for the process of laying out the circuit in VLSI.

c. Block Cascade/Slice Design

This is a compromise design between bit-sliced and cascaded structure. Here, an n-bit MIN unit is divided into blocks of smaller bit units. Each block unit is bit-slice designed. For a block cascade design, the lower order block still depends on the preceding block comparison decision not unlike the bit cascaded structure. A block diagram for this structure is shown in Figure 8. The block-sliced design is defined such as each block output is indepenent of previous and succeeding blocks comparison decision. This requires comparison look ahead capability for each block. Using the same definition and notation as in the previous section, let G_i, C_i, Z_i and Y_i be the decision variables for the ith block and A_i and B_i its binary input and O_i its output. Note that the variables are all capitalized to distinguish them from the variables used in the bit-slice design. If we define

$$Z_i = 1 \text{ when } A_i = B_i$$

$$Y_i = 1 \text{ when } A_i < B_i$$

then

$$C_i = C_{i+1}Z_i \tag{4}$$

$$G_i = G_{i+1} + C_{i+1}Y_i \tag{5}$$

and the output is given by

$$O_i = A_i G_i + B_i \overline{G_i} \qquad (6)$$

We see that these equations are similar to equations (1), (2) and (3). To render each block independent, we apply the same principle as in the bit-slice design. This requires yielding Z and Y variables for each block simultaneously. The following equations [Ref. 4] generates Z and Y for each block.

$$Z_i = Z_{id+d-1} Z_{id+d-2} \cdots\cdots Z_{id}$$

$$Y_i = Y_{id+d-1} + Z_{id+d-1} Y_{id+d-2} + \cdots\cdots + Z_{id+d-1} Z_{id+d-2} \cdots Z_{id+1} Y_{id}$$

The comparison look-ahead circuit can then be derived by recursively applying equations (4) and (5). A block diagram of the block-slice structure is as shown in Figure 9.

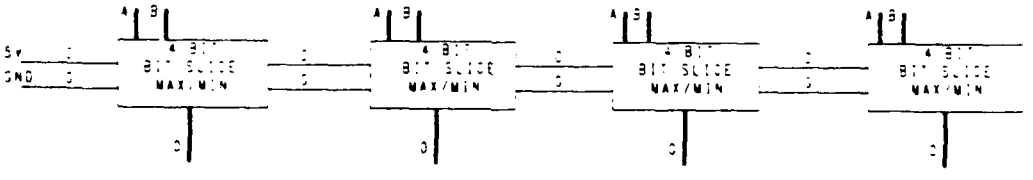


Figure 8 Block diagram of 16 bit block-cascade (4bit/block)

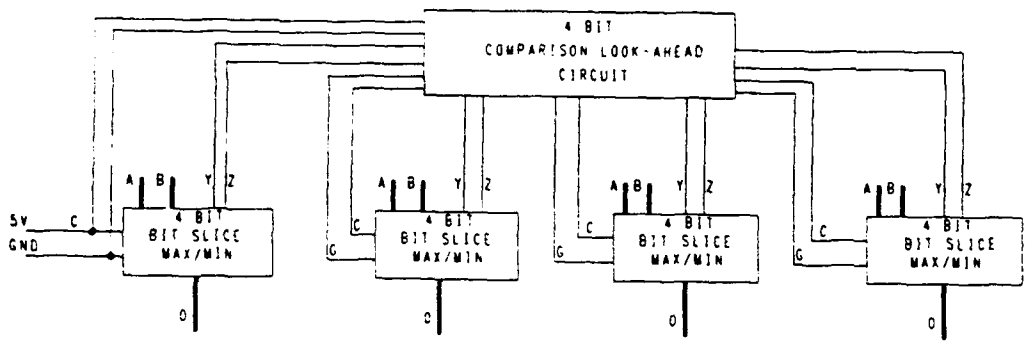


Figure 9 Block diagram of 16 bit block-slice (4 bit/block)

IV. VLSI IMPLEMENTATION PROCEDURES

In this chapter, we discuss the procedure of how the fuzzy operator units were implemented using VLSI CAD tools and the simulation results. The main steps were

1. Converting the Boolean logic gates circuit diagram to Complementary Metal Oxide Silicon (CMOS) transistors circuit diagram.
2. Layout of the transistor circuit diagram using MAGIC.
3. Functionality test of the layout using RNL.
4. Timing simulations using SPICE.

The circuits that were implemented and simulated are as follows:

- Bit Serial.
- 2 bit cascade and 2 bit bit-slice.
- 4 bit cascade and 4 bit bit-slice.
- 8 bit cascade and 8 bit bit-slice
- 16 bit cascade and 16 bit bitslice.
- 16 bit block cascade (4 bit/block).
- 16 bit block cascade (8 bit/block).
- 16 bit block-slice (4 bit/block).

A. CMOS TRANSISTOR CIRCUIT

The VLSI circuit for the various designs were implemented using CMOS technology. The circuits from the previous chapter were converted to their equivalent transistor circuit by directly substituting each logic gate with its equivalent CMOS transistor circuit. Figures 10, 11, 12 and 13 shows OR, AND, NOT and EXCLUSIVE-OR equivalent CMOS transistor circuit respectively. [Ref. 15] gives a detail account on these circuits.

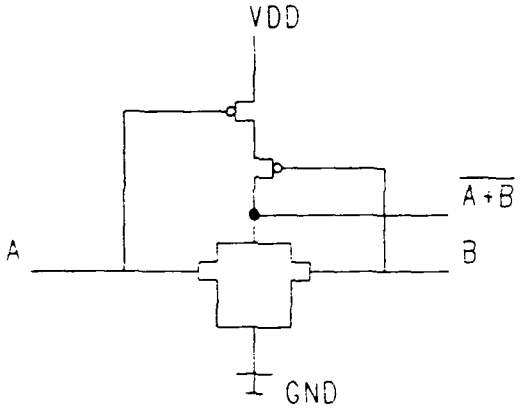


Figure 10 NOR gate equivalent CMOS transistor circuit.

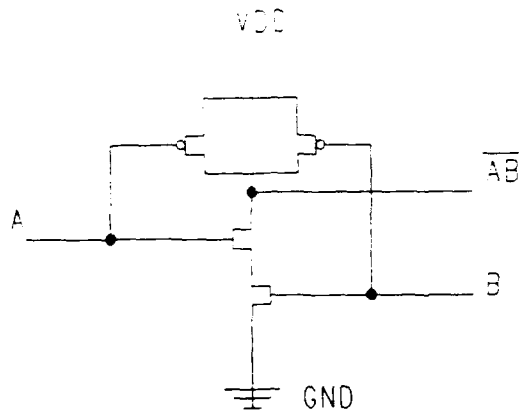


Figure 11 NAND gate equivalent CMOS transistor circuit.

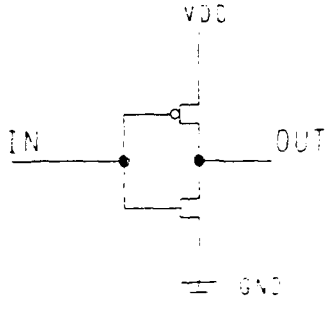


Figure 12 NOT gate equivalent CMOS transistor circuit

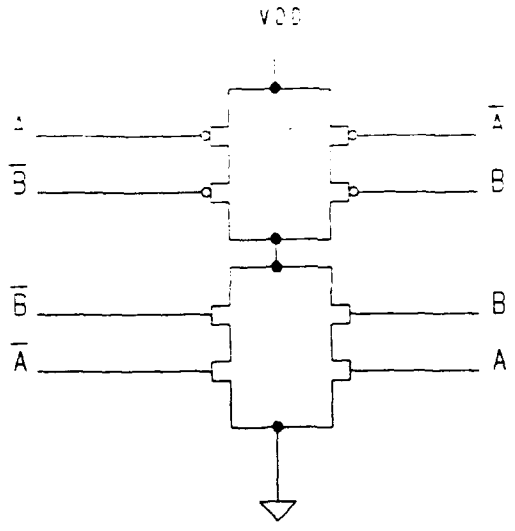


Figure 13 Exclusive-Or equivalent CMOS transistor circuit

B. VLSI CAD TOOLS

The CAD tools used in this thesis were Magic, RNL and SPICE. The following sections give a brief overview on these CAD tools.

1. Magic

Magic is an interactive CAD tool layout editing system for large-scale custom designed Metal Oxide Silicon (MOS) integrated circuits. It provides different technologies for custom-designed circuits. It was developed by faculty and students at the University of California, Berkeley in 1983. [Ref. 5] . Some of the features in Magic are:

- Uses rectangular or Manhattan style layouts.
- Contains user interactive layout editing operations.

- Checks layout for design rule violation automatically and informs user interactively.
- Contains a hierarchical circuit extractor to convert graphical layout to a file that contains information about sizes and shapes of transistors, connectivity, resistance and nodes capacitance of the layout. This file is needed in running simulation tools.

2. RNL

This is a timing logic simulator for digital MOS circuits. It is an event-driven simulator. It uses a simple resistance-capacitance model of the circuit. It estimates node transition times and the effect of charge sharing. RNL can be used for circuit verfication as well as timing measurements.

For simulation, RNL requires two sets of information:

- a description of the network to be simulated.
- a command to control simulation run.

The network to be simulated can be described using Magic layout or a netlist file (a textual schematics). Command input to RNL can be entered interactively or in batch mode. Command input includes initializing the network and applying stimulus signals. [Ref. 6]

3. SPICE

SPICE is an acronym for Simulation Program with Integrated Circuit Emphasis. It was developed at University of California, Berkeley. It is a general purpose simulation

program for nonlinear dc, nonlinear transient, and linear ac analyses.

Circuits are described to SPICE through an input file, which lists each circuit element (resistor, capacitor, inductance, voltage/current source, semiconductor device) and indicates how each is connected using node numbers. The file may also contain lines which designate the frequency of sources, temperature, the types of analyses to be done and how the analysis results are to be presented.

To analyze the circuit, SPICE first uses Kirchoff's current law to create a system of equations in which the voltages at each node are the unknowns and the admittance of each branch connecting two nodes is the known quantities. This group of equations is then made into an admittance matrix. The Newton-Raphson method is then used to solve this matrix. Various models for semiconductor devices are inherent in SPICE, the user has only to provide the model's parameter. The SPICE used in the thesis is version 2G6. [Ref. 7]

C. VLSI CIRCUIT LAYOUT OF FUZZY OPERATOR UNIT

All of the operator units layout were done using Magic. Some designs, such as bit-cascade and block-cascade which features regularity in its layout, use Magic first to generate the basic building blocks (lower level cells). Then Coordinate Free LAP (CFL) facilities are used to assemble (interconnect) these cells into the desired final form. CFL is a library of

subroutines written in C to facilitate the construction of VLSI circuit layouts. In-depth information about CFL can be found in [Ref. 6]. Designs with no regularity features had to be laid by hand. This includes block-slice and bit-slice designs. Magic layout for the various designs are given in the Appendix.

D. LAYOUT VERIFICATION

Layout verification is performed using RNL. Using the Magic extract command the layout information of the circuit was extracted into a file with a .ext extension. This file contains information about the circuit's transistor nodes and interconnection as well as its internodal capacitance and resistance.

Ext2sim program, part of UCB CAD tools, is then used to convert the .ext file produced by Magic to a flat circuit representation which is a requirement for simulation. This produces a file with a .sim extension.

Before we can run a RNL timing simulation, the .sim file must go through RNL preprocessor's PRESIM to convert it into a binary network file. The converted circuit layout is now ready for simulation. The result of running PRESIM also indicates the total number of transistors used in the circuit.

The verification process was done interactively after invoking RNL. A few input samples was presented to the circuit and the output was then verified accordingly. For example, for

an 8-bit MAX circuit, if A = 10011001 and B=11010001 then the output was verfied to be O = 11010001. Note that the time interval between each simulation steps has to be large enough to handle any propagation delay inherent in the circuit for the output to be verified correctly.

E. TIMING ANALYSIS

As stated in the previous section, in order to run SPICE we require an input file that describes the circuit. This is done through the use of sim2spice, another UCB CAD tool, that converts the .sim format file to a SPICE format. This file will only contain the list of transistors and capacitors. The transistors model parameters and simulation information has to be provided. [Ref. 16] gives more detail on how to do this.

SPICE simulation was only done on the following circuits

- INV
- serial design
- 1, 2, 4 bit cascade MAX/MIN
- 2, 4 bit-slice MAX/MIN

Due to memory limitation, SPICE was not used to simulate the other designs. RNL timing simulations was used to estimate these design timing performance. The SPICE simulation was run on ISI machines.

F. RESULTS

Table 2, Table 3 and Table 4 are the experimental results for the various fuzzy logic operator designs.

1. VLSI Estate Area

Table 2 shows the VLSI area used by the various designs. These areas may not reflect an optimum area since the layout process is an art by itself which depends very much on individual doing the layout. However to some degree the tabulated area may give us some means of making estate area comparisons. The table indicates that serial design uses the least area while bit-slice uses the most.

TABLE 2. LAYOUT VLSI ESTATE AREA

Design	Area in Scalable CMOS
Serial	57,120
n bit INV	893*n
n bit cascade	(24128*n) + 3248
16 bit cascade	389,296
2 bit-slice	94,640
4 bit-slice	190,986
8 bit-slice	528,700
16 bit-slice	1,790,019
16 bit block cascade (4 bits/block)	674,176
16 bit block cascade (8 bits/block)	1,123,332
16 bit block slice (4 bits/block)	1,584,435

2. Transistor Count

Table 3 gives the total number transistors that make up a particular design. We observe that bit-serial design uses the least number of transistors. Block-slice design used more transistors than bit-slice for a 16 bit unit.

TABLE 3. TRANSISTOR COUNT

Design	Number of transistors
Serial	98
n bit INV	2*n
n bit-cascade	58*n
16 bit-cascade	928
2 bit-slice	108
4 bit-slice	232
8 bit-slice	528
16 bit-slice	1312
16 bit block cascade (4 bits/block)	964
16 bit block cascade (8 bits/block)	1076
16 bit block slice (4 bits/block)	1358

3. Timing Analysis

To obtain the estimate of the speed of operation, we use the RNL timing simulation result as the basis of comparison among the various designs. SPICE results could not be used since we were only able to run simulation on designs of 4 bit cases. Table 4 shows the RNL and SPICE time delay measurements respectively.

The speed of operation is measured as the delay that occurs before the output of the operator unit become valid. We do this by changing the input bits and observing when the output bits settle to the correct result. To obtain the maximum ouput delay, for example for a 4-bit MAX unit, first we set input A = '0000' and input B = '0001'. This causes output O ='0001'. Then set A = '1000'. Since this is a MAX unit, output bit position 0 will change to a '0' since A is now greater than B. We measure the delay it takes for output bit 0 to change from a 1 to a 0. We then reset A = '0000' and measure the delay for bit 0 to change back to a '1'. We average these two delay measurements for our final result.

Notice that the delay measurements between RNL and SPICE are different. RNL, which is a switch-level timing simulator, uses the resistor-capacitor(RC) delay associated with each gate. This RC delay is estimated from the layout but does not include the detailed capacitance of junction and routing. This may explain the difference in the results.

The advantage of using RNL is that we can make a quick and rough estimation of the circuit's behavior. The mesurements may not however be used as the actual unit delay. SPICE is a much more accurate device simulation but takes a much longer time to run.

TABLE 4. RNL DELAY MESUREMENTS

Design	Time Delay
Serial	35 ns
INV	1.5 ns
1 bit MAX/MIN	2.5 ns
2 bit cascade	3.2 ns
4 bit cascade	4.8 ns
8 bit cascade	8.0 ns
16 bit cascade	14.3 ns
2 bit-slice	2.6 ns
4 bit-slice	3.2 ns
8 bit-slice	4.5 ns
16 bit-slice	7.3 ns
16 bit block cascade (4 bits/block)	8.6 ns
16 bit block cascade (8 bits/block)	7.7 ns
16 bit block slice (4 bits/block)	8.8 ns

TABLE 5. SPICE DELAY MEASUREMENTS

Design	Time Delay
1 bit MAX/MIN	14.2 ns
2-bit cascade	16.6 ns
4-bit cascade	24.3 ns
2 bit-slice	15 ns
4 bit-slice	17.6 ns

V. CONCLUSIONS

In the thesis we have implemented the basic fuzzy operation units i.e., the MAX, MIN and INV using VLSI. Various designs of MAX and MIN were investigated and the following conclusions were made,

- Bit-serial designs use the least VLSI estate area and transistors but the operation is slow compared to the other designs. It requires about 35ns to complete a 1 bit comparison. A better circuit design may have achieved better results.
- MAX and MIN uses the same number of transistors and VLSI estate area for the various designs. The main difference between these two circuits is in the magnitude comparator circuit.
- Adopting the fuzzy numbering system used in the thesis, the INV unit is only composed of inverters. This simplifies the design and the unit delay is independent of the number of bits used. However it should be noted that this numbering system does not have the representation for decimal 0.5. But as the number of bits used increases, this should not pose any problem.

Taking 16 bit as the basis of comparison for the four design schemes used in the MAX/MIN operator, the following are observed,

- Bit-cascade uses the least VLSI area whereas bit-slice takes up the most area. Block slice used nearly as much area as the bit-slice. Area-wise block-cascade seem to be the compromise. Here the number of bits per block affect the area used. 8-bit per block used twice as much as 4-bit per block.
- In terms of the least number of transistors used, bit-cascade or block-cascade(4-bit/block) is the better design.

- For timing performance, bit-slice design yields the better result.

On the basis of these observations, we may conclude that a fast fuzzy operation can be achieved if the number of transistors and VLSI area used does not matter. The alternative is to have a slower operation. Between the block-cascade and block-slice, there seem to be very minor differences in the timing performance but they vary greatly in the VLSI area and transistors used. Thus, one may conclude that a compromise design between the extreme of bit-slice and bit-cascade is the block-cascade or block-slice. Still between these two designs, one may choose one or the other depending on performance desired and the cost factor.

Further work may be carried out in using these designs as the basic building blocks for a fuzzy controller or fuzzy inference engine.

APPENDIX

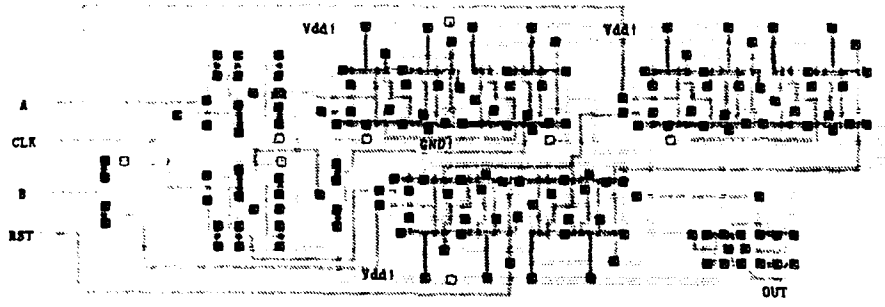


Figure A.1 Bit Serial Layout

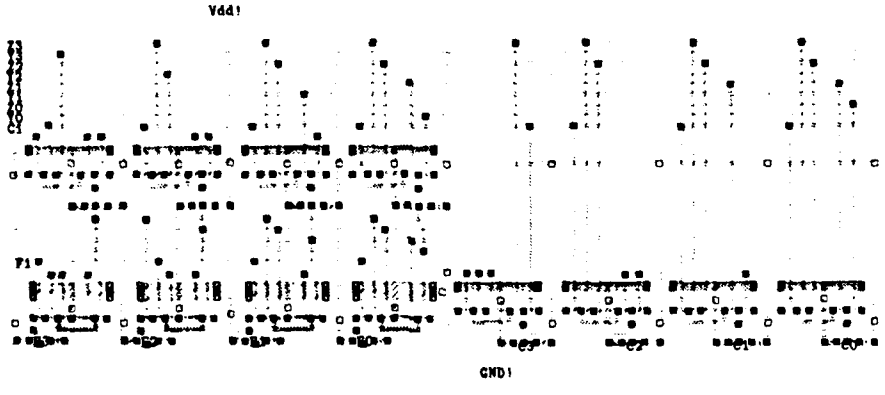


Figure A.2 Comparison look-ahead layout

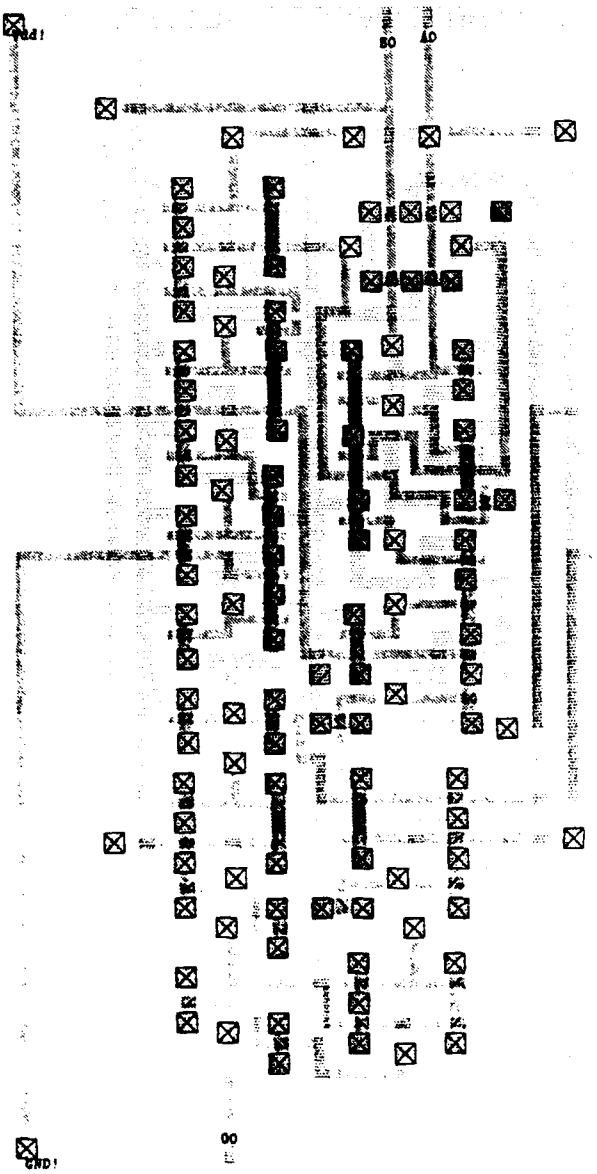


Figure A.3 1-bit MAX Operator Layout

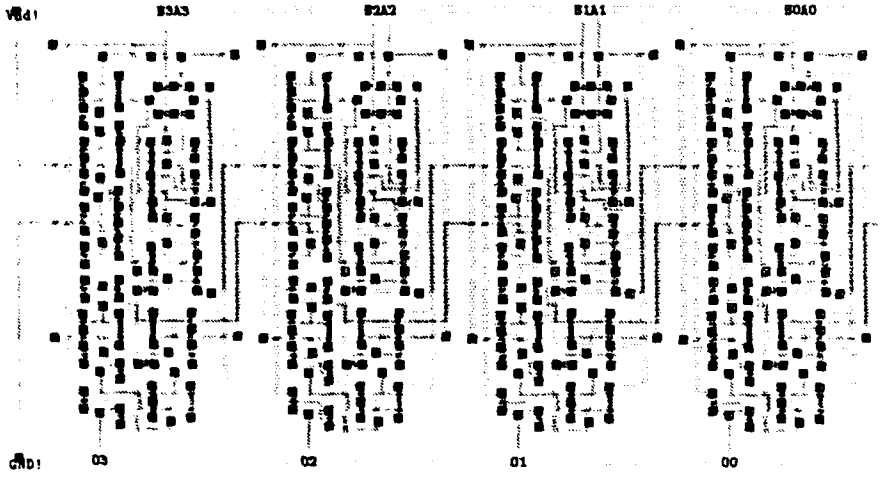


Figure A.4 MAX 4 bit-casacde Layout

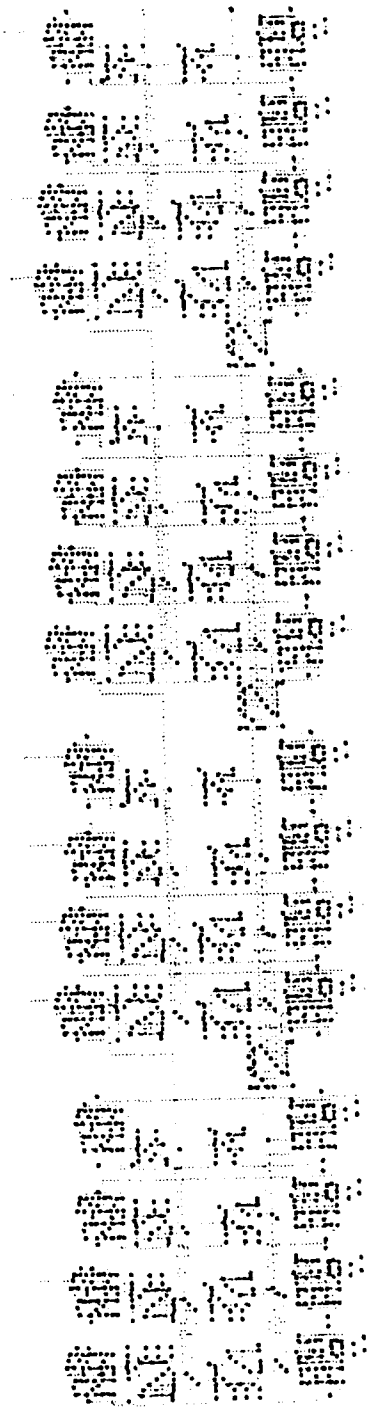


Figure A.5 16 bit Block-Cascade (4 bit/block) Layout

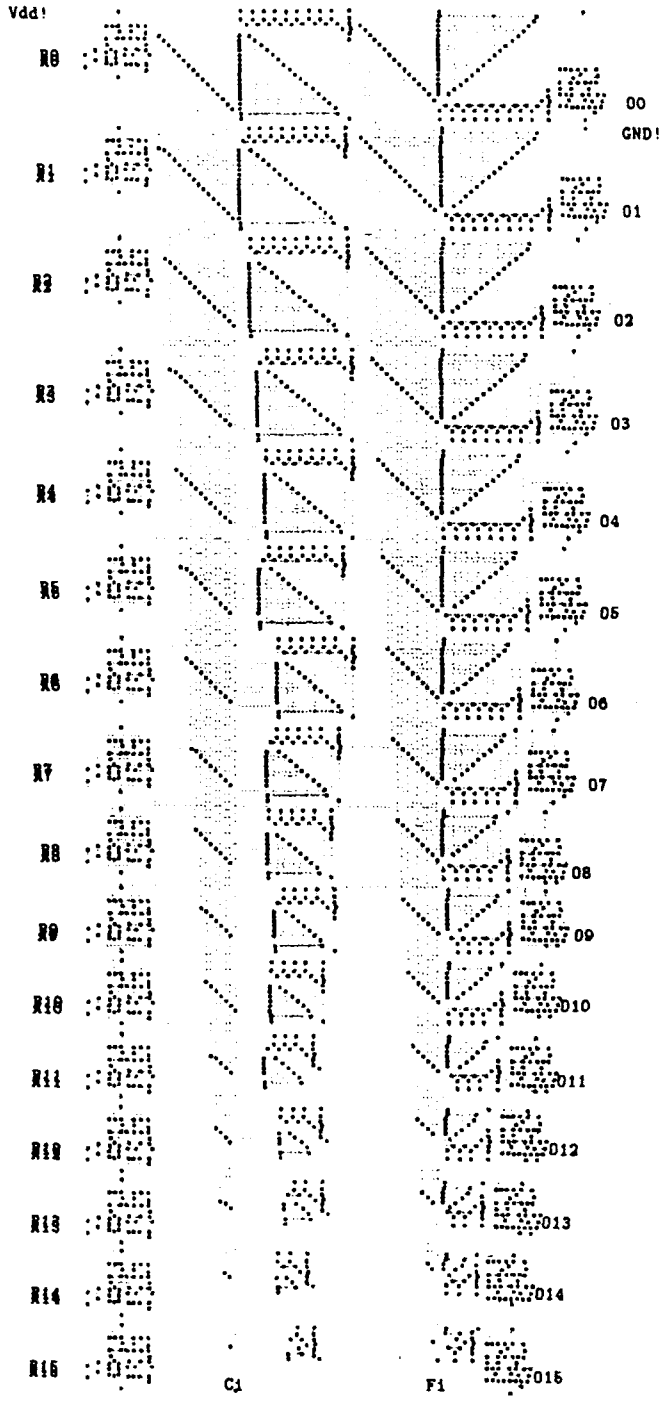


Figure A.6 16 bit-slice MAX Layout

LIST OF REFERENCES

1. L.A. Zadeh, "Fuzzy Sets," *Information and Control*, Vol. 8, pp. 338-353, 1965.
2. David K. Kahaner, Walter J. Freeman and Armando Freitas da Rocha, "Fuzzy Logic," *Scientific Information Bulletin*, Vol. 16, No.1, pp. 41-54 , Jan-Mar 1991.
3. J.T. Johnson, "Fuzzy Logic," *Popular Science*, pp. 87-89, July 1990.
4. Jia-Yuan, Han and Supreet Singh, "Comparison Look-Ahead and Design of Fast Fuzzy Operation Unit," *IEEE Multi-valued Logic Conference*, pp. 121-125, 1990.
5. University of California, Berkeley, Report No. UCB/CSD 86/273, *1986 VLSI Tools: Still More Works by the Original Artists*, by Walter S. Scott, Robert N. Mayo, Gordon Hamachi and John K. Ousterhout, December 1985.
6. NW Laboratory for Integrated Systems, TR # 88-09-01, *VLSI Design Tools Reference Manual Release 3.2*, Department of Computer Science, University of Washington, September 1988.

7. A. Vladimirescu, Kaihe Zhang, A.R. Newton, D.O. Pederson, and A. Sangiovanni-Vincentelli, "SPICE User's Guide," *UW/NW VLSI Release 2.1*, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.

8. L.A. Zadeh, "Fuzzy Logic," *Computer*, pp. 83-92, April 1988.

9. L.A. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," *IEEE Transactions Systems Man Cybernetic*, SMC-3, pp.28-45, 1973.

10. Abraham Kandal, and Samuel C. Lee, *Fuzzy Switching and Automata: Theory and Application*, Crane Russak and Company Inc., 1979.

11. E.H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with Fuzzy Logic Controller," *International Journal Man-Machine Studies*, Vol.7, pp. 1-13, 1975.

12 Masaki Togai and Hiroyuki Watanabe, "A VLSI Implementation of a Fuzzy Inference Engine: Toward an Expert System on a Chip," *Information Sciences*, Vol.38, pp. 147-163, 1986.

13. W.J.M. Kickert and H.R. Van Nauta Lemke, "Application of a Fuzzy Controller in a Warm Water Plant," *Automatica*, Vol.12, pp. 301-308, Pergamon Press 1976.

14. Kevin Self, "Designing with Fuzzy Logic," *IEEE Spectrum*, pp. 42-44, November 1990.

15. Neil Weste, and Kamran Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*, Addison-Wesley Publishing Company, 1985.

16. Naval Postgraduate School, *ECE VLSI Lab Notes*.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor Chyan Yang, Code EC/Ya Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Professor Jon T. Butler, Code EC/Bu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Jabatanarah Latih Department Tentera Laut Kementerian Pertahanan 50634 Kuala Lumpur, Malaysia	1
7.	Lt Cdr Ismail bin Dewa 958, Jalan Teratai 12 Taman Marida, Senawang 70450 Seremban, Negeri Sembilan Malaysia	1