THE UNIVERSITY OF MANITOBA

DECOMPOSITION TECHNIQUES FOR NAND CIRCUITS

by

D. M. MILLER

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

WINNIPEG, MANITOBA

OCTOBER, 1973

Abstract

A decomposition algorithm for the design of practical multilevel combinational switching circuits composed of two-input NAND gates is developed. The generalization to two-input gates of any type is, as shown, quite straightforward. The theoretical background required is presented, and from this, a new theory of two-place decomposition is developed. The algorithm is compared with previous algebraic, geometric, and decomposition methods. Several sample results are discussed, and a number of improvements and extensions proposed.

Acknowledgements

I wish to express my gratitude to:

Dr. J. C. Muzio whose teaching first stimulated my interest in logic design, and whose assistance was invaluable in the preparations of this thesis;

Dr. D. P. Kerr and Prof. D.A.Young for their careful review of this work;

Miss Eileen Robbins for her excellent preparation of the typed manuscript;

To my wife, Maureen, for her concern and patience, I am always grateful.

Table of Contents

Chapter 1

Introduction

Switching circuit theory is the study of the mathematical models of circuits made up of two-state, or bistable, devices. The term 'switching circuit' originates with the switch which was the first practical bistable device in use. Railway signaling, and automatic telephone exchanges were the first major applications of switching circuits. Today, switching circuits are used widely, from vending machines to digital computers. The computer industry has quickly become the largest single user, and has been responsible for most of the advances in both design and technology.

Initially, switching circuits were constructed by the heuristic application of several ad hoc design techniques. The first mathematical model was due to Shannon [36] who used a two-valued Boolean algebra to describe the behaviour of relay contact networks. Since that time, the use of algebraic simplification techniques in the design of switching circuits has been widely explored. These techniques are the most commonly used and we shall present the fundamental methods later.

A second approach is based on the correlation between switching circuits and mappings of point set geometry. Some positional notation is used to represent the possible configurations of the input variables, and the corresponding functional value. Topological techniques are then used to combine certain input configurations to achieve a reduced representation of the function, and, consequently, a simplified circuit realization.

Both these approaches result in minimal, or nearly minimal, AND/OR circuit realizations, and are quite adequate for relay networks. Practical circuits composed of electronic switching elements generally

have several levels of gating, and new design techniques must therefore be developed for the design of circuits of this type.

An electronic switching element, or gate, may be viewed as a black box with a certain number of inputs, and an output which is some binary combination of the inputs. Gates are characterized by the Boolean function describing the output in terms of the inputs. The most common are the NAND and NOR functions. A multilevel circuit constructed of gates may thus be viewed as a realization of a function as a composition of several simpler functions.

The theory of functional decomposition has been developed to handle representations of this form. Much theory has been presented on this subject, and several decomposition algorithms for circuit design have been developed. The principle result of this thesis is an efficient algorithm for the design of practical multilevel combinational switching circuits composed of two-input NAND gates. This algorithm is shown to be more useful than previous design techniques.

Chapter 2 introduces the algebraic approach using the fundamental work of Quine [30], [31]. An alternative method [41], which appears to require less computation, is also presented. In Chapter 3, the cubical calculus due to Roth [32] is presented. Several algorithms based on this calculus are discussed. These are representative of the geometric approach to the design of circuits. The cubical calculus will also form the basis of the notation and algorithms of subsequent chapters. The discrepancy between theory and practical circuit design is discussed in Chapter 4. The solution to these design problems is partially solved by an algorithm due to Su and Nam [39] which is therefore included. The theory of decomposition is introduced in Chapter 5. These results

form the basis for the development of our practical desing algorithm in Chapter 6. The results of an implementation of this algorithm are presented in Chapter 7 and several interesting improvements and future developments are considered.

The original development of this work is contained in Chapters 6 and 7 , the preceding chapters being a review of previous techniques. Details of implementation and much of the subsequent work based on these techniques has been excluded as this discussion was intended to be a background to the development of our decomposition algorithm. These results are well documented in the references, and in any of several books on switching theory.

Chapter 2

The Minimization Methods of Quine and Zissos

1. Introduction

The correlation between expressions of Boolean algebra and switching circuits was presented by Shannon [36] . Since that time, the use of the algebra in the design of circuits has been the subject of much research, and several systematic methods have been developed. A few definitions are universal to these techniques.

i) A 'variable' is any lower-case letter.

ii) A 'literal' is a variable or a negation of a variable.

iii) A 'fundamental formula' is a conjunction of literals in which no variable appears twice. A single literal may also be a fundamental formula.

iv) A 'normal form' is a disjunction of fundamental formulae. Each fundamental formula in the normal form is called a clause. A single fundamental formula may also be a normal form. (Both fundamental formulae and normal forms will be represented by upper-case letters. The context will clarify which is meant.)

v) A fundamental formula, A , is said to 'subsume' another fundamental formula, B , if, and only if, all the literals of B are among the literals of A .

vi) A fundamental formula, A , is called a 'prime implicant' of a normal form G , if, and only if, A implies G , and A subsumes no shorter formula which implies G . (A fundamental formula implies a normal form if, and only if, the normal form is true whenever the fundamental formula is true.)

vii) A normal form is called 'developed' if all of the variables in the normal form appear in each of the clauses. The clauses of a developed normal form are termed 'minterms'.

Most of the methods which have been developed depend on the basic research of Quine, who has presented both a tabular [30] , and an algebraic method [31] for determining the simplest normal form representation of an expression. Quine's methods are representative of this approach to the problem, and are thus presented below.

Zissos [41] has introduced an algebraic approach to the same problem which does not follow Quine's technique. This method is non-exhaustive and should require less work than previous methods, but this has not been established. Zissos' method is presented and compared with the methods of Quine.

2. The First Method of Quine

2.1 The Determination of Prime Implicants

The initial specification of the function must be given in developed normal form. A systematic application of the identity

$$Aa + A\bar{a} = A$$

is then used to find the prime implicants. Each pair of clauses is examined for those which differ by a single negation sign. When such a pair is found, the common part of the two clauses is added to the formula, and the original clauses are check marked.

In the expression $pqr + pq\bar{r} + \bar{p}\bar{q}r$, for example, the fundamental formulae pqr and $pq\bar{r}$ differ by a single negation sign. The common term pq is thus added to the formula to give

$$p\check{q}r + p\check{q}\bar{r} + \bar{p}\bar{q}r + pq$$

The process is applied iteratively until no further terms can be added. The terms which remain unchecked are the prime implicants of the function.

2.2 The Minimal Normal Form Equivalent

The minimal normal form equivalent is the formula with the fewest terms, and, within that limitation, the fewest literals. It is easily shown [17] that each term of such an expression is a prime implicant. Quine has, therefore, developed a tabular technique for extracting the minimal normal form from the set of prime implicants of the function.

The columns of this table are labeled with the minterms, and the rows are labeled with the prime implicants. A cross is placed at each row and column intersection for which the minterm subsumes the prime implicant. The following rules are then used to extract the minimal normal form:

i) For any column which has only one cross, record the row heading of the row containing the cross. These prime implicants are essential to represent the function and are termed the 'core'.

ii) Delete from the table all rows which satisfy i), and all the columns which have crosses in any of these rows.

iii) If one column has crosses only in rows which a second column has crosses, the latter column may be deleted.

iv) Delete any column whose crosses have been lost after ii) and iii).

As an example, consider the normal form

$$pqr + p\bar{r} + pq\bar{s} + \bar{p}r + \bar{p}\bar{q}\bar{r}\bar{s}$$

which has the prime implicant table:

	$pqrs$	$pqr\bar{s}$	$pq\bar{r}s$	$pq\bar{r}\bar{s}$	$p\bar{q}rs$	$p\bar{q}r\bar{s}$	$\bar{p}qrs$	$\bar{p}qr\bar{s}$	$\bar{p}\bar{q}rs$	$\bar{p}\bar{q}r\bar{s}$	$\bar{p}\bar{q}\bar{r}\bar{s}$
pq	X	X	X	X							
qr	X	X					X	X			
$p\bar{r}$			X	X	X	X					
$\bar{p}r$							X	X	X	X	
$\bar{p}\bar{q}s$										X	X
$\bar{q}\bar{r}s$						X					X

Applying rule i) and rule ii) we obtain the table below and a core of $\bar{p}r$ and $p\bar{r}$.

	$pqrs$	$pqr\bar{s}$	$\bar{p}\bar{q}\bar{r}\bar{s}$
pq	X	X	
qr	X	X	
$\bar{p}\bar{q}s$			X
$\bar{q}\bar{r}s$			X

From rule iii) either column 1 or column 2 may be deleted. We arbitrarily choose column 1.

	$pqr\bar{s}$	$\bar{p}\bar{q}\bar{r}\bar{s}$
pq	X	
qr	X	
$\bar{p}\bar{q}s$		X
$\bar{p}\bar{r}s$		X

From this table it is clear the terms $pqr\bar{s}$ and $\bar{p}\bar{q}\bar{r}\bar{s}$ may be covered in any of four ways, and combining these with the core, the simplest normal form equivalents are:

$$p\bar{r} + \bar{p}r + pq + \bar{p}\bar{q}\bar{s}$$

$$p\bar{r} + \bar{p}r + pq + \bar{q}\bar{r}\bar{s}$$

$$p\bar{r} + \bar{p}r + qr + \bar{p}\bar{q}\bar{s}$$

$$p\bar{r} + \bar{p}r + qr + \bar{q}\bar{r}\bar{s}$$

As the number of terms and literals in all four forms is the same, there is no advantage in using one in preference to the others.

Much work has been done toward optimizing and extending Quine's fundamental process. Notably, McCluskey [19] has introduced a simplified notation for the clauses, and a systematic approach to the generation of the prime implicants which requires fewer comparisons. McCluskey has also introduced an expanded set of prime implicant table simplification rules which includes the deletion of a prime implicant which at any stage only has crosses in columns which are all covered by a second prime implicant.

Petrick [28] has introduced a method which, for any prime implicant table, yields the entire set of irredundant normal form equivalents. The minimal form is then easily found. While exact, the method is not readily programmable, and becomes exceedingly lengthy for a table with any more than a few crosses.

Quine's method has been extended to problems with 'don't care' conditions [17], and to the simplification of multiple output functions [3]. Improvements to these techniques have been suggested by Pyne and McCluskey [29], Ghazala [8], Necula [27], Morreale [24], Luccio [18], and Choudhury and Das [4].

3. The Second Method of Quine

3.1 The Generation of the Prime Implicants

Quine's second method [31] also involves the generation of the prime implicants, and subsequent extraction of the minimal normal form. The starting point, however, is any normal form expression and the computation involved is generally considerably less. The following rules are used to generate the prime implicants:

i) Delete the obvious superfluities; if one of the clauses subsumes another, it is deleted. Also, $a + \bar{a}B$ can be replaced by $a + B$, and $\bar{a} + aB$ can be replaced by $\bar{a} + B$.

ii) Add the consensus of two clauses to the expression.

If two clauses contain the same variable, negated in one and affirmed in the other, the consensus is the conjunction of the clauses with that variable removed. Otherwise, the consensus does not exist. For example, the consensus of $ab\bar{d}$ and $\bar{a}bc$ is $bc\bar{d}$.

Rule ii) is not applied if the consensus subsumes a clause in the expression as this would initiate an oscillation between rule i) and rule ii).

The process continues iteratively until neither rule can be applied, at which time the expression is a disjunction of all the prime implicants.

3.2 The Minimal Normal Form

A dispensing operation is used to extract the minimal normal form.

Definition - A prime implicant is dispensable if each minterm it implies is implied by at least one other prime implicant.

Dispensable clauses are bracketed. After examining the entire expression, the unbracketed terms form the core. Bracketed terms which imply the core are then deleted. Groups of bracketed clauses must then be examined for joint dispensability.

Definition - A group of clauses is jointly dispensable if, when the entire group is removed, each clause in the group implies the remaining expression.

For example, consider

$$ps + \bar{p}\bar{s} + \bar{q}t + prs + qr\bar{s} + pqrt$$

prs subsumes ps and may therefore be deleted, leaving

$$ps + \bar{p}\bar{s} + \bar{q}t + qr\bar{s} + pqrt \quad .$$

The consensus of ps and $qr\bar{s}$ is pqr which is subsumed by pqrt .

$$ps + \bar{p}\bar{s} + \bar{q}t + qr\bar{s} + pqr$$

Reapplying rule ii) gives

$$ps + \bar{p}\bar{s} + \bar{q}t + qr\bar{s} + pqr + (r\bar{s}t + prt)$$

$(r\bar{s}t + prt)$ are jointly dispensable as they were added after the last application of rule i) . This is self-evident as the clauses from which they were formed still appear in the expression. We find that $qr\bar{s}$ and pqr are individually dispensable. The core is thus simply

$$ps + \bar{p}\bar{s} + \bar{q}t$$

The above method applied to

$$ac + bc + c\bar{d} + \bar{c}d + \bar{a}\bar{b}\bar{c}$$

yields the solutions

$$\bar{a}\bar{b}\bar{c} + c\bar{d} + ad + bd + \bar{a}\bar{b}\bar{d}$$

$$c\bar{d} + \bar{c}d + ad + bd + \bar{a}\bar{b}\bar{d}$$

$$ac + bc + c\bar{d} + \bar{c}d + \bar{a}\bar{b}\bar{c}$$

However, a Karnaugh map yields a minimal solution of

$$ac + bc + \bar{c}d + \bar{a}\bar{b}\bar{d} \quad .$$

This problem can be avoided by using the prime implicant table techniques of Quine's first method.

4. The Minimization Method of Zissos

4.1 Introduction

Zissos and Duncan [41] have presented a method for finding the minimal normal form of a Boolean function. The principle advantage is that once a normal form equivalent is found, the expression is never expanded. The method has similarities to Quine's second method in that the developed normal form is not required, and the consensus of terms is used in the minimization process. Using Zissos' terminology, a normal form is a sum of products expression, and the consensus of two terms is their optional product. The product is optional so long as the parent terms, those terms from which the product was formed, remain in the expression.

A similar method, based on Zissos' preliminary research, has been developed by Knispel [16] . The latter has also looked at don't-care conditions, and the multiple-output problem.

4.2 The Irredundant Expression

The first step is to determine an irredundant sum of products expression of the function.

Definition - A Boolean expression is said to be irredundant if it contains no optional products or factors, that is, products or factors whose presence do not affect the value of the function.

Initially, the given expression is transformed into sum of products form using the distributive laws of Boolean algebra [6] . For example:

$$a(b + c) + \bar{a}b + (\bar{a} + b)c = ab + ac + \bar{a}b + \bar{a}c + bc$$

Let $H1$, $T1$, $T2$, and I be Boolean expressions. The theorem

Theorem 2.1

$$(H1 + T1 + I)\ (\overline{H1} + T2 + I) = H1T2 + \overline{H1}T1 + I$$

is used to avoid the generation of redundant terms. For example, direct application of the distributive law yields

$$(a + b)\ (c + \bar{b}) = ac + a\bar{b} + bc + b\bar{b}$$

which can be reduced to

$$a\bar{b} + bc \quad .$$

Choosing $H1 = b$, $T1 = a$, $T2 = c$, and $I = \phi$, theorem 2.1 would yield this result directly.

The distributive laws, and theorem 2.1 , are applied until all brackets have been removed. The identity $a \cdot \bar{a} = 0$ is used to remove any trivial terms, and the identity $A + AB = A$ to remove terms whose

truth value is implied by another term in the expression.

Once these first order redundancies have been removed, a systematic application of the optional products theorem

Theorem 2.2

$$aB + \bar{a}C = aB + \bar{a}C + BC$$

is used to remove redundant terms. The description of this technique is taken from Zissos and Duncan [41] .

Assuming the products are arranged from left to right in ascending order of size, proceed as follows:

i) The first variable in the first product is selected, and the remainder of the expression is scanned for a product that contains the complement of the selected variable. When such a product is found, form an optional product using theorem 2.2 . The optional product is used to eliminate non-parent products and, or, to replace parent products. If a parent product has been replaced, insert the optional product at the beginning of the expression and repeat. If the optional product has not been used to replace a parent, it is discarded.

This process is repeated until all first level optional products have been generated.

ii) The process is repeated using higher level optional products.

The following examples demonstrate this step.

Example 2.1

$P = a + \bar{a}b + bc + \bar{a}\bar{b}d$

b - replaces parent $\bar{a}b$ and non-parent bc

$= b + a + \bar{a}\bar{b}d$

$\bar{a}d$ - replaces parent $\bar{a}\bar{b}d$

$= \bar{a}d + b + a$

d - replaces parent $\bar{a}d$

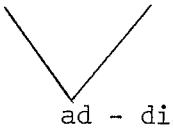
$= d + b + a$

Example 2.2

$P = ab + \bar{a}c + \bar{b}d + \bar{c}d$

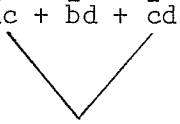
bc - discard

$= ab + \bar{a}c + \bar{b}d + \bar{c}d$



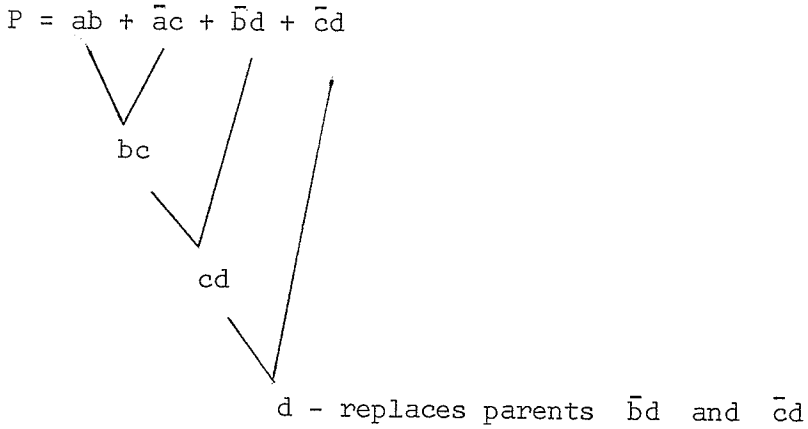
ad - discard

$= ab + \bar{a}c + \bar{b}d + \bar{c}d$



$\bar{a}d$ - discard

Generating higher level optional products



$$= d + ab + \bar{a}c$$

The reduction process terminates when every optional product has been tried. At this point, the expression is irredundant.

4.3 The Minimal Expression

It does not always follow that an irredundant expression is minimal. The irredundant expression

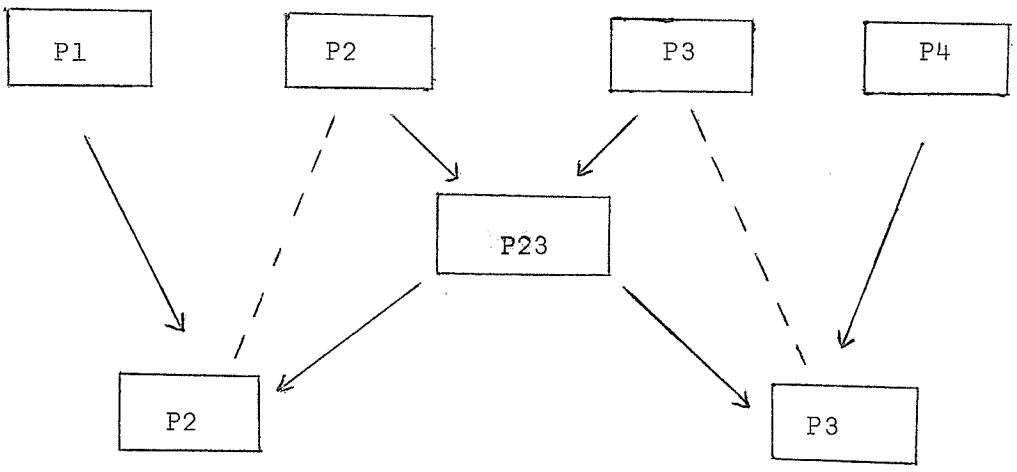
$$P = a\bar{c} + ab + \bar{a}c + \bar{a}\bar{b}$$

is equivalent to

$$P' = a\bar{c} + bc + \bar{a}\bar{b}$$

and is, therefore, not minimal. A minimization step must be performed to ensure a minimal result.

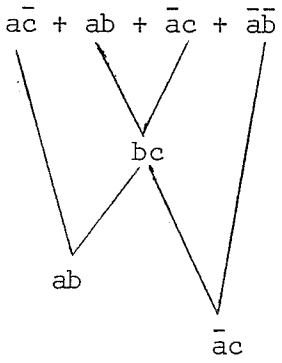
The condition for a simplification at this stage can be diagrammed as



A necessary criterion for this condition is that there exist a set of four terms in which:

i) some variable, say a, appears or can be made to appear, at least twice in its true form, and at least twice in its inverted form,

and ii) two other variables, say b and c, are each present at least once in their true form, and at least once in their inverted form.



If a simplification of this type is found, the reduction process is reapplied. The two steps are performed alternately, until the minimization criterion is not satisfied. At that time, the expression is

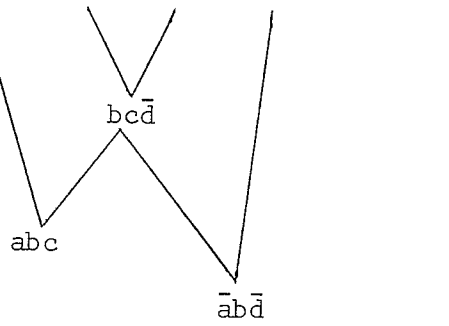
believed to be minimal.

A complete example follows:

$$P = abc + abd + \bar{a}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{d} \quad .$$

An examination of the optional products indicates this expression is irredundant. We thus proceed with the minimization step.

$$P = abd + abc + \bar{a}b\bar{d} + \bar{a}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{d}$$



$$P = abd + bc\bar{d} + \bar{a}\bar{c} + \bar{b}\bar{d}$$

It was necessary to replace $\bar{a}\bar{d}$ by $\bar{a}b\bar{d} + \bar{a}\bar{b}\bar{d}$ in order to achieve the correct structure. Conditions for detecting this structure and for proper literal substitution are given by Zissos and Duncan.

The reduction step is reapplied

$$P = abd + bc\bar{d} + \bar{a}\bar{c} + \bar{b}\bar{d}$$

$c\bar{d}$ - replaces parent $bc\bar{d}$

$$P = abd + c\bar{d} + \bar{a}\bar{c} + \bar{b}\bar{d}$$

The minimization step cannot be applied and as expected, the above expression is minimal.

The validity of this minimization criterion has not been established. Zissos and Duncan have, however, solved several examples without detecting any inadequacies.

5. Discussion

The principle advantage of Quine's first method is that it is a strictly mechanical procedure which, when carried to its conclusion, will yield both the minimal and near-minimal normal form solutions. A major drawback, however, is the required specification of the problem in developed normal form. For some problems, the work involved in finding the prime implicants is fantastic.

Quine's second method generally requires less work in finding the prime implicants. Also, the developed normal form is not required. The extraction of the result by the dispensing operation does not always yield a minimal result, and in general, the prime implicant table of the first method is used.

The use of prime implicants may lead to serious problems. Fridshal [7] has presented the following minimum upper bounds on the number of prime implicants.

Number of Variables	Number of P.I.
2	2
3	6
4	13
5	32
6	92
7	218
8	576
9	1,698
10	4,300
11	11,000

Certain examples will be encountered which are formidable even to the faster computers.

Zissos has avoided this problem. His method does not generate all the prime implicants, but, it would appear, only those required for a minimal solution. Several questions still remain unanswered. The term 'higher order optional product' is nowhere defined and it is not known if this includes the optional product of optional products. Secondly, no provision has been made for the same optional product being generated from different parents, and considerable redundant computation could occur. Most importantly, the validity of the method has not been established and the strict minimality of the result cannot be assured.

All three methods result in normal form, or sum of products, solutions. The circuits derived directly from these results are impractical and must be transformed to NAND or NOR circuits. Even when this is done, certain design problems are encountered. A systematic procedure for obtaining practical NAND circuits from a normal form result will be presented in chapter 4.

Chapter 3

The Cubical Calculus of Roth

1. Introduction

The Harvard Computational Laboratory [9] developed a chart technique for the simplification of switching circuits. This was of little practical interest as the solution of an n variable problem involved the examination of 2^{2^n} entries. A more compact method was suggested by Montgomerie [23], and developed by Vietch [40] and Karnaugh [12]. In particular, a systematic application of Karnaugh's simplification rules always yields a minimal normal form result. The method, unfortunately, does not lend itself to machine computation, and the work involved in manually solving a problem in more than five or six variables is tedious and prone to error, if not totally impossible.

These methods are geometric approaches to the simplification problem, and even their modest success seems to indicate that further investigations along this line might prove fruitful.

Roth [32] has identified the correlation between Boolean functions and mappings of combinatorial topology. A convenient positional notation has been developed and operators consistent with Boolean algebra have been defined. Among the results which were presented are the cubing, star, and sharp algorithms which are equivalent to the work of Quine.

In this chapter, a non-rigorous treatment of the calculus is presented together with the algorithms mentioned. This calculus forms the basis of the notation and most of the theory of subsequent chapters.

2. The Cubical Complex of a Boolean Function

A Boolean function $f(a_1,a_2,\ldots,a_n)$ may be represented by a disjunctive normal form expression, and , in fact, by any one of several such expressions. There is, however, a finite set of clauses which imply the function, and from which the above expressions are formed. This set is the complex of the function.

Let Q^n represent the set of all fundamental clauses of n variables. Each such term, T , may be represented by an ordered set of n symbols $\{t_1,t_2,\ldots,t_n\}$, where $t_i = 1$ if a_i appears affirmed in T , $t_i = 0$ if a_i appears negated, and $t_i = x$ if a_i does not appear. Such an n-tuple is termed a cube. A cube containing no x's is termed a vertex. There is a one-to-one correspondence between the fundamental clauses formed from n variables and the set of all n - co-ordinate cubes. We associate this set of cubes with Q^n and therefore term it the n-cube.

Clearly, the complex of a Boolean function f of n variables may be represented by a subset, denoted $K(f)$, of the n-cube. $K(f)$ is the cubical complex of the Boolean function.

Let Z_2 be the set $\{0,1\}$ and Z_2^n be the n^{th} order Cartesian product of Z_2 . Let f be a mapping of X onto E , denoted $f: X \to E$, where $X \subseteq Z_2^n$ and $E \subseteq Z_2$. The inverse mapping $f^{-1}(1)$ yields a set $Y \subseteq X$ such that $x \in Y \Rightarrow f(x) = 1$. There is a one-to-one relation between the set of all such mappings and the set of all Boolean functions of n-variables. Also, there is a one-to-one relation between the set Y and the set of all minterms of a Boolean function.

Definition - Two cubes, a and b, are opposed in their i^{th} co-ordinates, denoted a_i and b_i, if one of a_i or b_i equals one, and the other equals zero.

Definition - Two cubes are adjacent if they are opposed in exactly one co-ordinate.

Two adjacent vertices may be expressed as a cube equal to the vertices with the opposed co-ordinate replaced by an x. The vertices are the faces of the cube, and the operation of forming the cube is termed cubing. The minterms corresponding to these vertices may be replaced by a single term equal to the minterms with the opposed variable removed. Similarly, two r-dimensional cubes, cubes containing exactly r x's, which are opposed in one co-ordinate may be represented by an $r + 1$ dimensional cube. The cubing of two cubes over the i^{th} co-ordinate is simply a geometric application of the theorem

Theorem 3.1

$$a_i B + \overline{a_i} B = B$$

It has been shown by Quine [30] that beginning with the developed normal form, the set of all fundamental clauses which imply the function may be found by an iterative application of theorem 3.1. It follows that given $Y = f^{-1}(1)$ the cubical complex $K(f)$ may be found by repetitive cubing operations.

An elementary co-cycle of $K(f)$ is a cube which is not a face of a higher order cube of $K(f)$. The corresponding term T is a prime implicant at it implies f, but no other term T' exists such that

$T \Rightarrow T' \Rightarrow f$.

As $K(f)$ may be determined from Y , and from this set the subset of elementary co-cycles may be extracted, the complex notation is sufficient to implement the first method of Quine. This application was presented by Roth [32] , and in a somewhat different form by McCluskey [19].

The consistency of the cubical complex notation with the disjunctive normal form has led to the development of a calculus. The principle results are presented below.

3. A Calculus of Boolean Complexes

A fundamental operation of the algebra is the conjunction of Boolean expressions. In the calculus of complexes, therefore, an intersection operator must be defined. The intersection of cubes, denoted $a \cap b$,is defined by the table:

$a_i \cap b_i$

a_i \ b_i	0	1	λ
0	0	ϕ	0
1	ϕ	1	1
x	0	1	x

and the rule:

$a \cap b = \phi$ if for any i , $a_i \cap b_i = \phi$

else, $a \cap b = (a_1 \cap b_1, a_2 \cap b_2, \ldots, a_n \cap b_n)$

where $a = (a_1, a_2, \ldots, a_n)$

$b = (b_1, b_2, \ldots, b_n)$.

The intersection of the complexes A and B is a complex C whose member cubes are $c_k = A_i \cap B_j$ for all $A_i \varepsilon A$, and $B_j \varepsilon B$. If A represents the Boolean function f,, and B represents the Boolean function g , then the intersection $A \cap B$ represents the function $h = f \cdot g$.

Consider the complexes;

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & x \\ 0 & x & x \end{pmatrix} \qquad B = \begin{pmatrix} x & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The complex $C = A \cap B$ is

$$C = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

which may be reduced to

$$C = \begin{pmatrix} x & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad .$$

The intersection of cubes may also be used in the detection of subsuming terms in a disjunctive normal form expression. Recall that given terms T_1 and T_2, T_1 subsumes T_2, denoted $T_1 \subseteq T_2$, only if all the literals of T_2 appear in T_1. The equivalent criterion in the calculus being $a \cap b = a$ where a is the cube representing T_1 and b is the cube representing T_2. The significance of subsuming terms is that they do not affect the value of the function. Subsuming cubes are thus removed from a complex.

4. The Cover of a Cubical Complex

A cover of a cubical complex A is a subset B of A such that for every vertex $v \subseteq a \in A$, $v \subseteq b \in B$, and conversely, for all $v \subseteq b \in B$, $v \subseteq a \in A$. The problem of finding a cover with a minimal

number of cubes of highest dimension is a restatement of the synthesis problem of Quine.

The more general problem which shall be treated is finding a set C of cubes of K which cover a subcomplex L of K. If K, C, and L define Boolean functions f, g, and h respectively, then C is a normal form expression for g such that $h \Rightarrow g \Rightarrow f$. C is a K - cover of L and the vertices in $(K - L)$ are don't-care conditions.

4.1 The Cubing Algorithm

The first algorithm due to Roth is a systematic application of cubing to find Z the set of elementary co-cycles, or prime-implicants, of K. As expected, this method parallels the Quine-McCluskey algorithm. Roth does not, however, introduce the partitioned list techniques of Morreale [24]. Groupings are made by free variables, those whose co-ordinate value is x, and some work is saved. A crude upper bound of $3^n 2^{n+1}$ single co-ordinate comparisons is established for a problem of n variables. Both methods suffer from similar problems; the most serious being the required specification of the function as 0-cubes, or minterms.

4.2 The *-Algorithm

Roth has introduced an algorithm similar to the second method of Quine. Here the consensus of two terms is equivalent to the *-product of the corresponding cubes. This product is defined by the table:

$a_i * b_i$	b_i: 0	1	λ
0	0	y	0
a_i 1	y	1	1
x	0	1	x

and the rule;

$a * b = \phi$ if $a_i * b_i = y$ for more than one i

else $a * b = (\psi(a_1 * b_1), \psi(a_2 * b_2), \ldots, \psi(a_n * b_n))$

where $\psi(0) = 0$, $\psi(1) = 1$, $\psi(x) = x$, $\psi(y) = x$.

The case when $a_i * b_i \neq y$, for all i , is degenerate and $a * b = a \cap b$. Such terms are ignored as they subsume their parents. The following simplified statement of Roth's algorithm is due to Dietmeyer [6] .

Let C be a given cover of a complex K .

i) For all cubes C_i and C_j of C if $C_i * C_j \neq \phi$, then add $(C_i * C_j)$ to C .

ii) Remove all $C_k \in C$ such that $C_k \subseteq C_\ell \in C$, $k \neq \ell$.

iii) Continue until

a) $C_i * C_j = \phi$ for all C_i , $C_j \in C$

or b) all $C_i * C_j \neq \phi$ are such that $(C_i * C_j) \subseteq C_k \in C$.

An efficient implementation of this algorithm may be achieved by requiring that before the algorithm begins, any subsuming cubes of C are removed. Step ii) then only requires that the new cube $(C_i * C_j)$ be compared with each $C_k \in C$. Further, if a C_k is found such that $(C_i * C_j) \subseteq C_k$, then no more comparisons need be made. It is also evident that if $(C_i * C_j)$ is degenerate, then it may be immediately rejected as $(C_i * C_j) = (C_i \cap C_j) \subseteq C_i$ and $(C_i * C_j) = (C_i \cap C_j) \subseteq C_j$.

Roth has established that the above algorithm always yields the subset Z of elementary co-cycles of K. Although Z is a cover of K, it is unfortunately not, in general, minimal. The cubing algorithm is a restricted case of this algorithm as it requires that $(C_i * C_j) \supset C_i$, and $(C_i * C_j) \supset C_j$ before the product is added to C. This restriction requires that the original elements of C all be 0-cubes. Although the result is the same, the *-algorithm generally requires far less computation.

4.3 The # Extraction Algorithm

Once Z the set of elementary co-cycles of K is found, the remaining problem is to extract a minimal subset C of Z such that C covers L. The initial step is to determine the cubes of Z which are essential to the cover. This set E is termed the set of K-extremals and has the property that if $C_i \in E$ then there exists at least one vertex $v \subseteq C_i$ such that $v \not\subseteq C_j$ for any $C_j \in Z$, $i \neq j$. The basis of this operation is the sharp product, denoted $a \# b$.

Let T_1 be a product term with cubical representation a. Also let T_2 be a product term represented by b. The expression h represented by $(a \# b)$ is such that $h = T_1 \cdot \bar{T}_2$. h, in general, is not a single product term and is represented by the disjunction of a set of product terms. Consider

$$T_1 = a\bar{b}d \quad \text{and} \quad T_2 = \bar{b}\bar{c}de$$

The result defined by $T_1 \# T_2$ is

$$T_1 \cdot \overline{T_2} = a\bar{b}d\ \overline{(\bar{b}\bar{c}de)}$$

$$= a\bar{b}d\ (b + c + \bar{d} + \bar{e})$$

$$= a\bar{b}cd + a\bar{b}d\bar{e}$$

A form of differencing operation is thus performed. The sharp product of cubes is defined by the table:

a_i # b_i	b_i: 0	1	x
v_i: 0	ε	ϕ	ε
1	ϕ	ε	ε
x	1	0	ε

and the rules:

i) $a \# b = a$ if $a_i \# b_i = \phi$ for any i

ii) $a \# b = \phi$ if $a_i \# b_i = \varepsilon$ for all i

else iii) $a \# b$ is the complex whose cubes are

$$(a_1, a_2, \ldots, \overline{b_i}, \ldots, a_n)$$

for all $(a_i \# b_i) \in \{0,1\}$.

For the above example T_1 is given by 10x1x and T_2 is given by x001 1 . The #-product is thus

$$10x1x \# x001\,1 = \varepsilon\varepsilon 1\varepsilon 0$$

which defines the complex

$$1011x$$

$$10x10 \quad .$$

The #-product of a complex $A = \{A_1, A_2, \ldots, A_n\}$ and a cube b consists of all the cubes given by

$$(A_1 \# b), (A_2 \# b), \ldots, (A_n \# b)$$

If A has subsuming terms removed, then the complex $A \# b$ has no subsuming terms.

The #-product of complexes A, and $B = \{B_1, B_2, \ldots, B_m\}$ is given by

$$A \# B = \{\ldots\{\{A \# B_1\}\#B_2\}\ldots\#B_m\} \quad .$$

Again, if the complex A has subsuming terms removed, then the product $A \# B$ has no subsuming terms.

The following distributive and pseudo-commutative properties were presented by Roth:

$$(a + b)\#c = (a \# c) + (b \# c) \quad \ldots\ldots \text{ distributive}$$

$$(a \# b)\#c = (a \# c)\#b \quad \ldots\ldots \text{ pseudo-commutative} \quad .$$

It can be shown that $Z_i \varepsilon Z$ is a K-extremal elementary co-cycle of K if, and only if,

$$(\ldots((\ldots(((Z_i\#Z_1)\#Z_2)\#\ldots\#Z_{i-1})\#Z_{i+1})\#\ldots)\#Z_n \neq \phi \quad .$$

This test, by the pseudo-commutative property of the #-product, is independent of the ordering of the cubes of Z. An exhaustive test of all the cubes of Z will yield E the set of K-extremals.

Once E is found, a minimal cover for the remaining uncovered cubes of L must be found. An algorithm for this process has been described by Roth. While still utilizing the calculus, the method is based on theory outside the scope of this discussion. Criterion for eliminating certain co-cycles from consideration, and the solution of cycling problems by branching are presented. In this way, the method is similar to the prime-implicant table techniques of Quine.

The initial results of the calculus were presented in [32]

with further development in [33] . Dietmeyer [6] has presented a simpler formulation with emphasis placed on the use of the calculus in computer programs for the design of circuits.

Chapter 4

The Design of NAND Circuits by Factoring

1. Introduction

At present, the vast majority of switching circuits are constructed of solid state devices known as gates. Each gate has a fixed number of binary inputs, and a single output which is a binary combination of the inputs. Gates which realize the functions NAND and NOR are most commonly used for several reasons.

First, both NAND, and NOR are functionally complete; that is, any Boolean function can be expressed entirely in terms of NAND, or NOR operators. Second, the final stage of a NAND, or NOR gate is a transistor which acts as an amplifier, thus avoiding the signal depletion problems found in circuits which use only passive components. Third, several gates of a single type can be constructed on a single integrated chip. These chips are extremely dependable, require little cooling, and are relatively inexpensive. A single chip with four two-input, three three-input, two four-input, or one eight-input NAND gate costs about twenty-five cents when purchased by the hundred. Finally, the space and power requirements are far less than what is required for relay or vacuum tube circuits.

We will restrict our discussion to NAND circuits. The techniques developed easily extend to NOR circuits by the principle of duality. Other types of circuits could also be handled by similar methods, but certain alterations would be required due to the different design problems encountered.

2. Three-Level NAND Circuits

Let $T_1, T_2, T_3, \ldots, T_n$ be fundamental formulae. The normal form

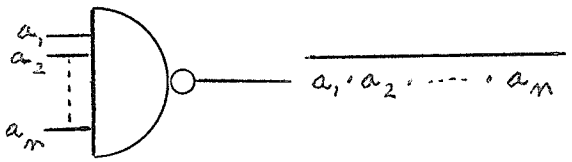
$$T_1 + T_2 + T_3 + \ldots + T_n$$

can be rewritten

$$\overline{\bar{T}_1 \cdot \bar{T}_2 \cdot \bar{T}_3 \cdot \ldots \cdot \bar{T}_n}$$

by applying De Morgan's theorem [6]. This expression can be realized by a circuit constructed entirely of NAND gates.

The negated fundamental formulae, and their negated conjunction, can each be realized by a single NAND gate. We shall use the symbol



to represent a NAND gate. The number of inputs will vary with its use.

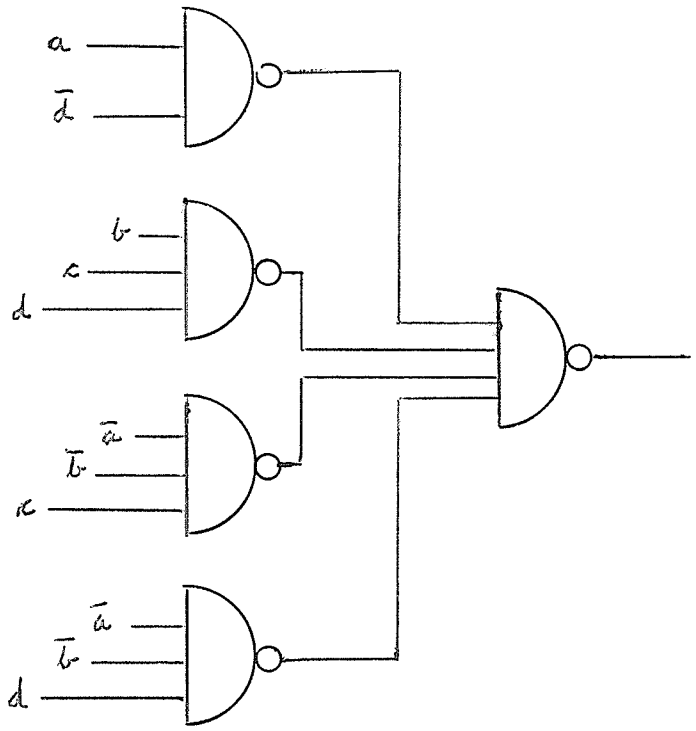
For example, the normal form

$$a\bar{d} + bcd + \bar{a}\bar{b}c + \bar{a}\bar{b}d$$

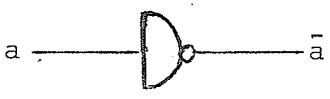
may be written

$$\overline{\overline{(a\bar{d})} \cdot \overline{(bcd)} \cdot \overline{(\bar{a}\bar{b}c)} \cdot \overline{(\bar{a}\bar{b}d)}}$$

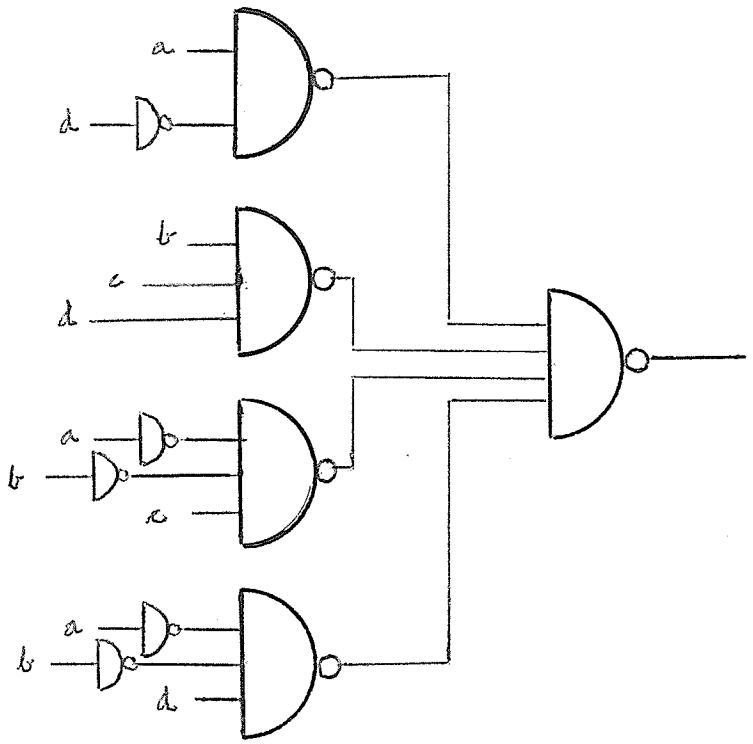
and realized by the circuit



Quite often, the negated inputs are not available and must be realized explicitly within the circuit. In practice, a single transistor is used as the inverter. At the schematic level this is equivalent to a single-input NAND gate, and we shall represent inverters as



The realization above becomes



Definition - The level of a NAND circuit is the maximum number of gates an input signal must pass through in order to reach the output.

The above circuit is a three-level NAND circuit. In general, the NAND circuit realized from a normal form expression, with at least one negated variable, is a three-level circuit.

3. Practical Design Criteria for NAND Circuits

NAND circuit design is subject to the criteria cost, fan-in limit, fan-out limit, and response time. Each of these is explained below.

Most commonly, the cost of a circuit is taken to be directly proportional to the total number of gate inputs within the circuit. This is quite reasonable as a single chip contains either two four-input, three three-input, four two-input, or one eight-input NAND gate. Thus, no matter which type is used, the cost of each gate input is quite nearly the same. What is not taken into account is the cost of any unused chip segments. If seventeen two-input NAND gates were required, five chips would be used, and three gates would be effectively wasted. There is the possibility that these gates could be used in nearby circuits, and even if they are not, the relatively small gate cost makes them insignificant.

The fan-in limit is the maximum number of inputs the largest gate in the circuit may have. This of course depends on the resources available and is therefore a variable.

There is a maximum amount of current which can be drawn from a gate without affecting its operation, and a minimum amount of current necessary to drive each of its inputs. Consequently, the number of gate inputs which can be driven by the output of a single gate is limited. The maximum number allowable is the fan-out limit, and once again depends on the resources available.

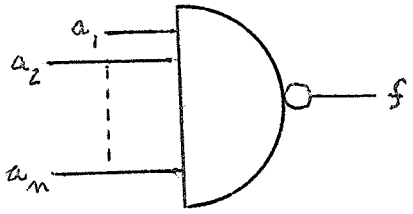
Each gate output requires a finite period of time to react to a change in its inputs. This response time is on the order of 15 ηs . The response time of the circuit is thus directly proportional to the level of the circuit. Because of the extremely high speed of the gates, response time is not critical and is often lengthened in order to solve other design problems.

A good circuit design thus takes into account fan-in and fan-out limits while simultaneously minimizing the cost and response time.

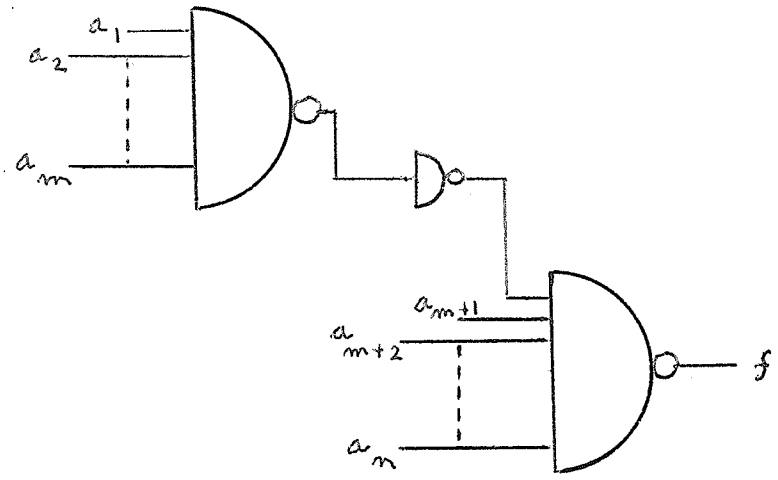
A technique known as factoring is often used in solving fan-in or fan-out problems.

4. Factoring and Cascaded NAND Gates

The most obvious method of solving a fan-in problem is demonstrated by the following diagrams:



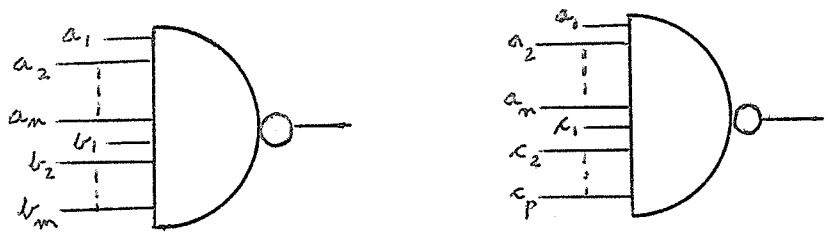
is equivalent to



Algebraically, this may be written

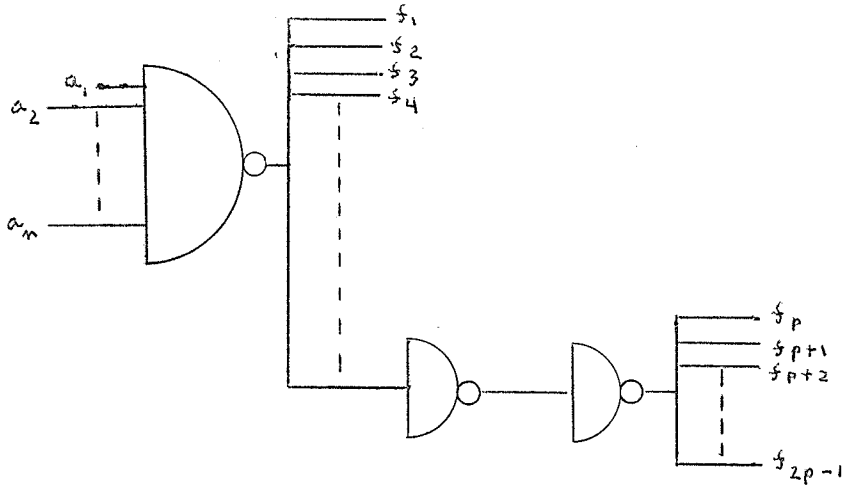
$$\overline{a_1 a_2 \cdots a_n} = \overline{\overline{(a_1 a_2 \cdots a_m)} a_{m+1} \cdots a_n}$$

Let two gates in a circuit be



and assume both exceed the fan-in limit. Any subset of the inputs could be brought back from each gate, but, clearly, it would be most advantageous to use $a_1, a_2, \ldots, a_n$ for both gates as only two additional gates would be required. Such sharing of common factors will greatly reduce the cost of the circuit.

Fan-out problems are solved using cascaded NAND gates. Let the fan-out limit be p. Two NAND gates can then be used to increase the fan-out to $2p - 1$.



Clearly, each fan-out problem is independent of the others and must be treated separately.

5. The Algorithm of Su and Nam

Su and Nam [39] have presented an algorithm for the solution of fan-in and fan-out problems in multiple-output multilevel NAND circuits. A systematic application of the factoring and cascaded NAND gates techniques is used. Particular emphasis has been placed on common factorizations, the largest common factor being used at each stage.

A modified cubical complex, called a function array, is used to specify the multiple-output function. In a similar manner to Bartee's method [3], a tag is appended to each input cube. This tag contains a single position for each output function. This position is a one if the fundamental formula represented by the cube implies the function, a zero if it does not, and the letter 'd' if it is a don't-care condition. A concise yet complete representation of several functions is thus achieved.

The first step in the algorithm is to apply a method due to Su and Dietmeyer [38] to find a connection array which completely describes a nearly minimal three-level NAND realization of the multiple-output function. This array consists of input and output parts. Each row of the input half represents a NAND gate realizing the negation of a fundamental formula, and each column of the output half represents a NAND gate collecting certain second-level gates to realize a particular function. The method involves a systematic assignment of the don't-care conditions to achieve optimum reduction, and for some problems may be excessive.

Once the connection array is found, it is used as a shorthand notation for the circuit and fan-in and fan-out problems are solved directly from it. The input half of the array is treated first.

The number of zeros or ones in each row is counted, and for those rows where this count exceeds the fan-in limit, the zeros and ones

are circled. Then, the number of zeros and number of ones in each column is counted. If the number of zeros exceeds the fan-out limit, the zeros are squared. Ones which violate the fan-out limit are also squared.

Let S be the submatrix of rows of the input array which have elements with circles or squares. The optimum factor corresponds to the submatrix S' of S which satisfies the conditions:

i) S' has at least two rows and two columns.

ii) S' has only one unique row.

iii) Of all S" which satisfy i) and ii), S' has the largest figure of merit (FM) where

$$FM = W_i \times N_i + W_0 \times N_0$$

W_i = input weight

W_0 = output weight

N_i = number of circles in S'

N_0 = number of squares in S'

W_i and W_0 are predetermined. A value of one was found to be most suitable for both.

Once the optimum factor has been found, a new input array which reflects this factorization must be formed.

The factor is recorded as a new row added to the input array. The co-ordinates which appear in S' assume their corresponding values while all others become x. The row is uniquely labeled and represents a two-level NAND cascade.

In addition, a column is added to the input array. For those rows which appear in S', this new column becomes one. For all other

rows, it is x . This column is assigned the same label as the new row, as it represents the NAND cascade as an input variable to later stages in the circuit.

Finally, as the inputs of the factor have been removed from their original positions, the co-ordinates of the input array which appear in S' are set to x .

This process proceeds iteratively until no further factors can be found.

The second part of the algorithm is concerned with fan-in and fan-out problems in the output array. The number of ones in each column is counted, and the ones circled in any column for which this value exceeds the fan-in limit. Similarly, the ones in rows for which the number of ones exceeds the fan-out limit are squared. An optimum factor is chosen in the same manner as for the input array, except that the conditions of a single unique row becomes a single unique column.

Once the factor is chosen, a column is added to the output array. This column is one for the rows of the chosen factor, and zero everywhere else. A unique label is used to identify this new cascade. A row with ones indicating the columns of S' is added and identified by the same label as the new column. The elements of the output array which appear in S' are set to d , and the process is repeated.

Once all fan-in and fan-out problems which result in common factors have been removed, any remaining problems are solved by cascaded NAND gates.

An example due to Su and Nam [39] will clarify the method.

Consider the connection array, and corresponding three-level NAND circuit, in Figure 4.1 . If the outputs, z_1, z_2, z_3, and z_4 are

FIGURE 4.1

	a_1	a_2	a_3	a_4	Z_1	Z_2	Z_3	Z_4
e_1	x	1	1	1	1	1	0	1
e_2	1	1	1	x	1	0	1	1
e_3	1	0	0	1	0	0	0	1
e_4	0	x	1	x	0	1	1	0
e_5	0	0	x	x	0	1	0	0
e_6	1	1	x	1	1	0	0	0
e_7	x	0	0	x	0	0	1	0

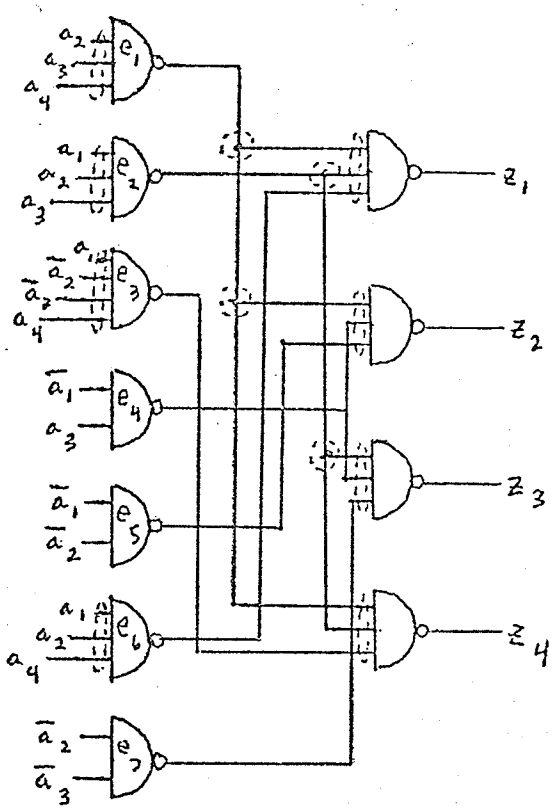


FIGURE 4.2

	a_1	a_2	a_3	a_4	Z_1	Z_2	Z_3	Z_4
		S_i'				S_o'		
e_1	x	1	1	1	1	1	0	1
e_2	1	1	1	x	1	0	1	1
e_3	1	0	0	1	0	0	0	1
e_4	0	x	1	x	0	1	1	0
e_5	0	0	x	x	0	1	0	0
e_6	1	1	x	1	1	0	0	0
e_7	x	0	0	x	0	0	1	0

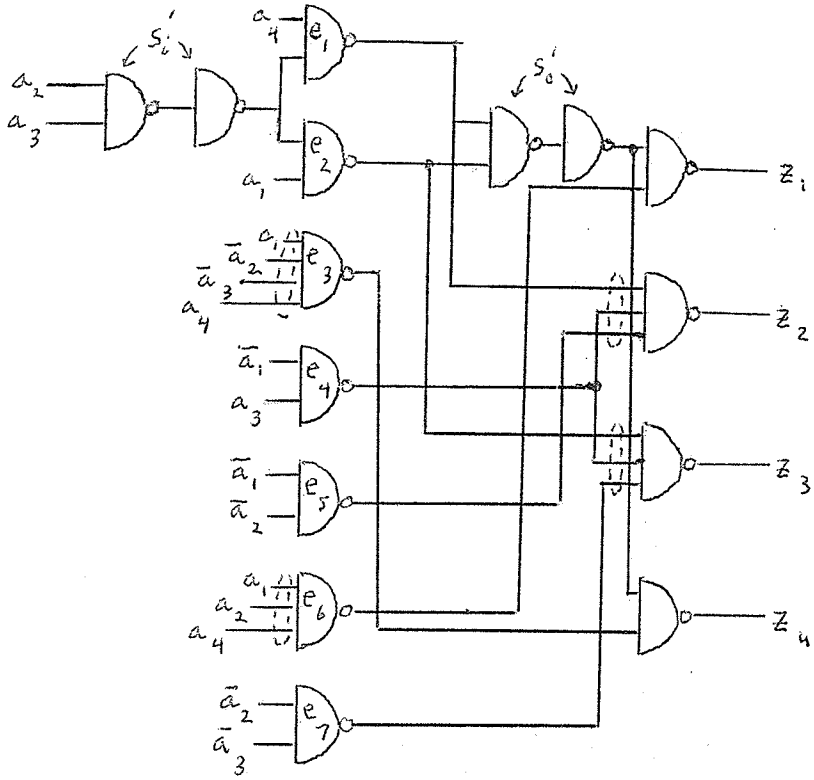
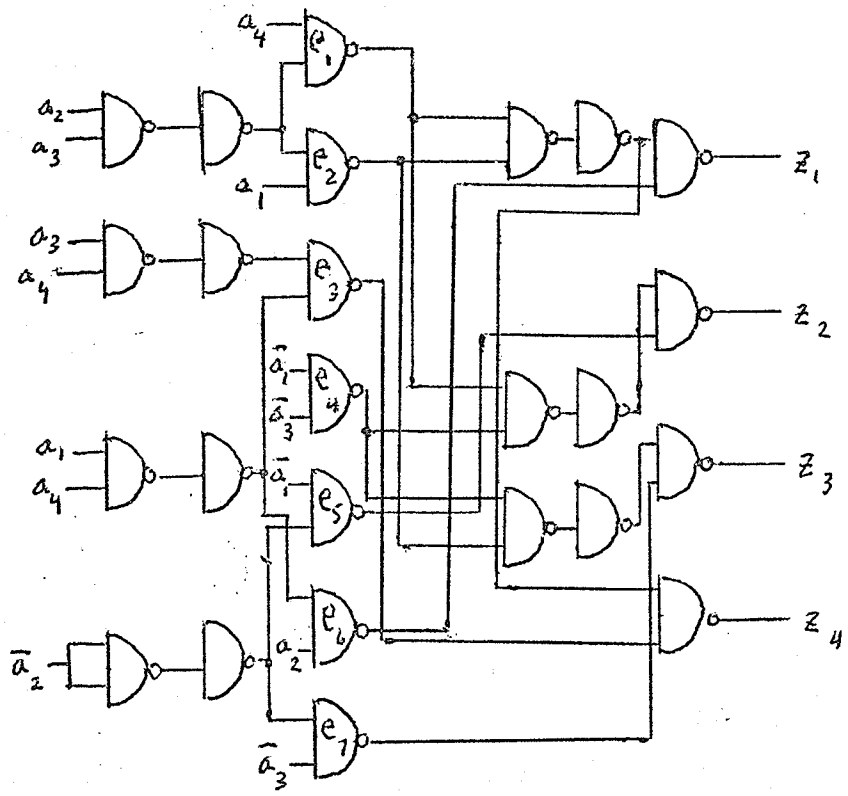


FIGURE 4.3

	a_1	a_2	a_3	a_4	S_i'	Z_1	Z_2	Z_3	Z_4	S_0'
e_1	x	x	x	1	1	d	1	0	d	1
e_2	1	x	x	x	1	d	0	1	d	1
e_3	1	0	0	1	x	0	0	0	1	0
e_4	0	x	1	x	x	0	1	1	0	0
e_5	0	0	x	x	x	0	1	0	0	0
e_6	1	1	x	1	x	1	0	0	0	0
e_7	x	0	0	x	x	0	0	1	0	0
S_i'	x	1	1	x	x	1	0	0	1	0 ← S_0'

FIGURE 4.4



to be realized using NAND gates with fan-in and fan-out of two, the broken loops indicate the problem areas in the circuit.

Performing both steps as described above, the optimum input and output factors, S_i' and S_0', are found. These factors together with the resulting circuit are shown in Figure 4.2 .

The connection array specifying the new circuit is found as described above. This array is presented in Figure 4.3 . The broken loops indicate the co-ordinates affected by the process.

The circuit resulting from a complete factorization is presented in Figure 4.4 .

Figure 4.5 shows a three-level NAND circuit for a 'two out of five' checker. This circuit is 'true' if, and only if, exactly two of its inputs are 'true'. A slightly modified circuit was used on the IBM 7090 computer.

Figure 4.6 shows a multilevel circuit found by applying Su and Nam's algorithm. NAND gates with a fan-in of two and unlimited fan-out were allowed. This circuit is slightly more expensive and, unfortunately, much more complicated. A far better multilevel realization will be presented in Chapter 7 .

6. Advantages and Disadvantages of the Algorithm

The principle advantage is the speed with which the algorithm can be implemented. Su and Nam have presented a ten-input, seven-output example which was completely factored in 2.4 seconds on a CDC 6400 . A statistical analysis has been performed on a random sample of several types of function. Gate count reductions of about 70 to 110% were obtained using the algorithm as opposed to solving each fan-in and fan-out problem

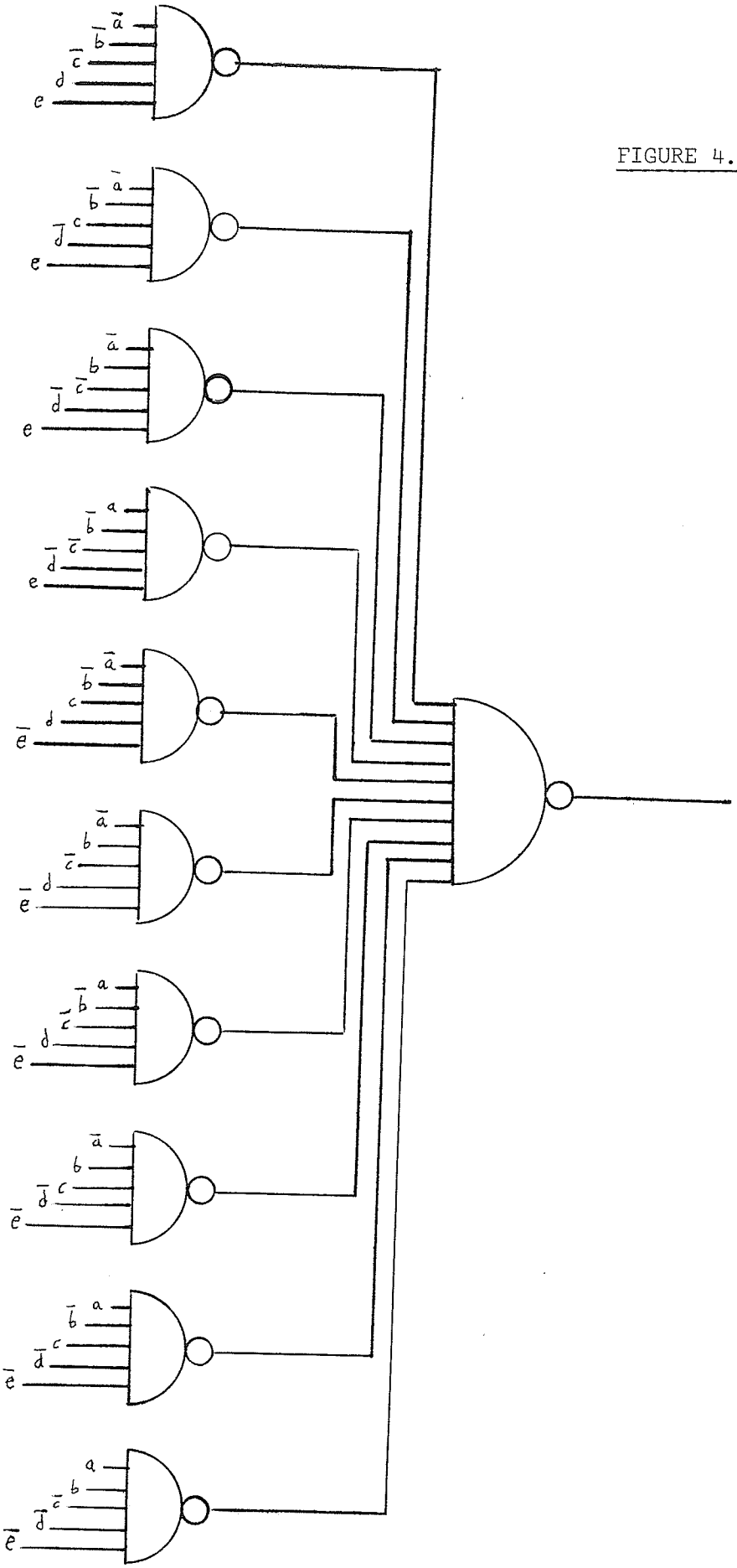
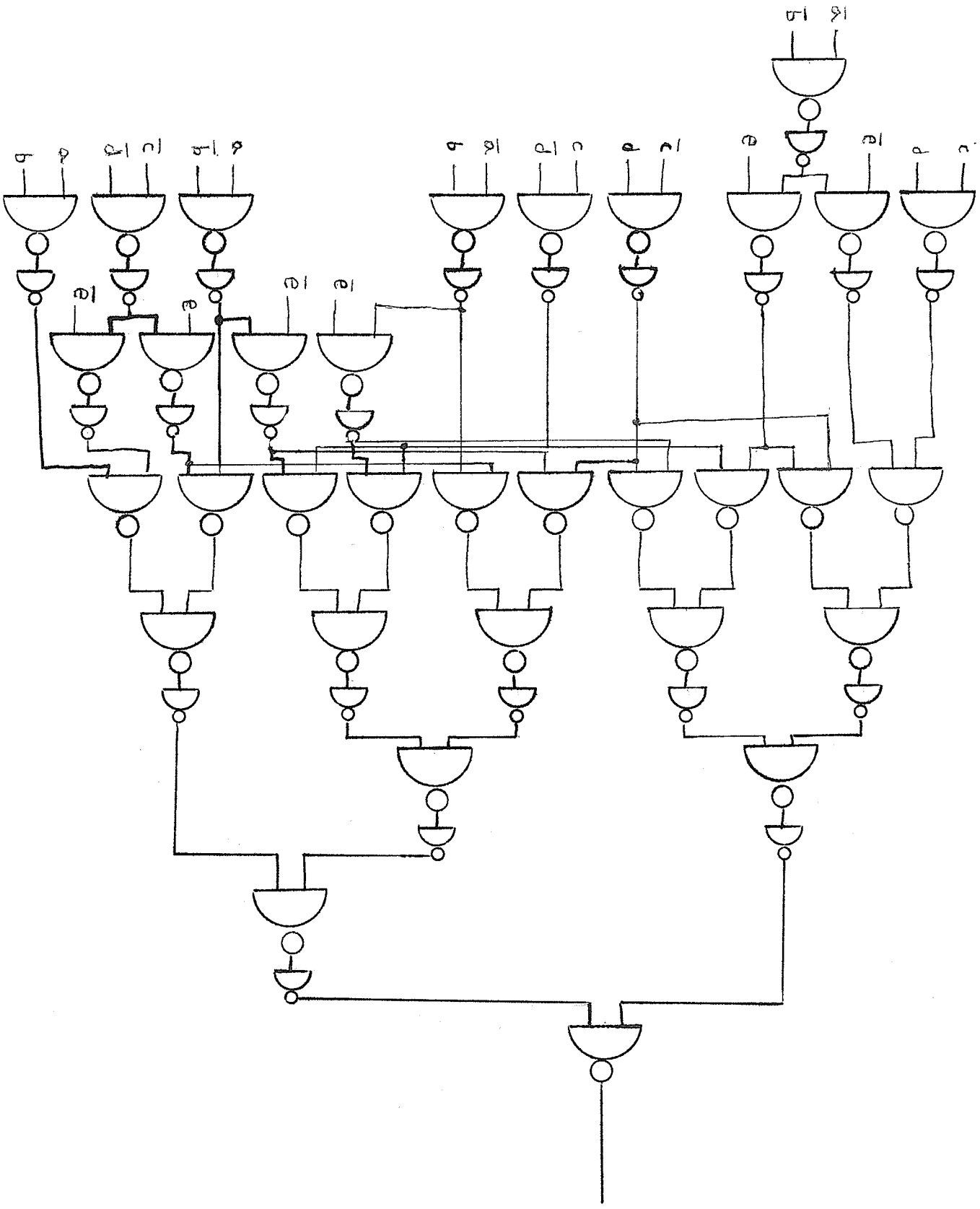

ā b̄ c̄ d e
ā b̄ c d̄ e
ā b c̄ d̄ e
a b̄ c̄ d̄ e
ā b c d ē
ā b c̄ d ē
a b̄ c̄ d ē
ā b c d̄ ē
a b̄ c d̄ ē
a b c̄ d̄ ē

FIGURE 4.5

FIGURE 4.6



separately. A final advantage is that the amount of work involved in multiple-output problems is not excessive.

The principle disadvantage of the algorithm is the requirement that a fan-in, or fan-out, problem exists before a factorization is performed. Gates which do no exceed the limits are not considered, and, therefore, certain factorizations which would further reduce the cost of the circuit may be missed. For example, suppose the following input array is to be factored with fan-in and fan-out limits of four and ten respectively.

	a_1	a_2	a_3	a_4	a_5	a_6
e_1	(1)	(0)	(1)	(1)	x	(1)
e_2	(1)	(0)	(1)	(0)	(1)	x
e_3	1	0	1	0	x	x
e_4	x	1	1	x	0	0
e_5	1	x	1	x	0	1

S'

The optimum factor is denoted S' . The row e_3 was not circled as it did not exceed the fan-in limit. S' is, however, a factor of e_3 and should be removed.

As a further example, assume the fan-in limit was increased to five. The factor S' would then be missed altogether.

The algorithm is implicitly assuming that all gates which do not exceed the fan-in limit have the same cost. This will, in some instances, lead to very costly circuits; particularly if the fan-in limit is high.

A second disadvantage is that from an algebraic point of view, the factoring of the expression representing the circuit is severely

restricted. Only product terms may be factored out, and the negation of a factor is never used.

The expression

$$ac + ad + bc + bd$$

is not reducible by this algorithm. However,

$$ac + ad + bc + bd = (a + b) \cdot (c + d) \quad .$$

The latter expression requires three two-input NAND gates, while the former requires four two-input and a single four-input gate.

A final disadvantage is that the input and output arrays are factored independently. The resulting circuits thus still are of a two stage nature. The first is a multilevel circuit which realizes the negations of the original fundamental formulae, and the second stage is a multilevel circuit realizing the outputs.

The algorithm is simply a systematic way of solving the problems encountered in implementing normal form solutions as practical circuits. We shall see in Chapter 7 that much better results are achieved by a completely different approach to the entire design problem.

Chapter 5

Decomposition

1. Introduction

Decomposition is the re-expression of a switching function of n variables as a composite of several functions, each depending on less than n variables. For example, the function

$$f(a,b,c,d) = ac + bc + ad + bd$$

may be written

$$f(a,b,c,d) = \gamma(\alpha(a,b),\beta(c,d))$$

where $\gamma(\alpha,\beta) = \alpha\cdot\beta$, $\alpha(a,b) = a + b$, and $\beta(c,d) = c + d$. Decompositions where each subfunction $\alpha,\beta,\gamma,\ldots,\delta$ can be realized by a primitive switching element are of particular interest as they represent multilevel circuit realizations of the original function. The problem is to find the decomposition corresponding to the 'best' circuit. Usually the 'best' circuit is the one with the lowest cost.

Ashenhurst [1] has introduced a theory of decomposition of totally specified Boolean functions, which has been compiled and extended by Curtis [5] . Simple disjunctive decomposition forms the basis of the Ashenhurst-Curtis theory.

Let A be the set of n - input variables $\{a_1,a_2,\ldots,a_n\}$, and let $f(A)$ be a Boolean function. Given $A_\lambda \subseteq A$, and $A_\mu \subseteq A$, where $A_\lambda \cap A_\mu = \phi$, and $A_\lambda \cup A_\mu = A$, f is said to have a simple disjunctive decomposition if, and only if, there exist Boolean functions α and g such that

$$f(A) = g(\alpha(A_\lambda),A_\mu) \quad .$$

The decomposition is termed simple as there is only a single α , and disjunctive as A_λ and A_μ are disjoint.

Ashenhurst has presented a criterion for the existence of a decomposition for a given $f(A)$, and partition of A . This involves the examination of decomposition charts [3Y] . Each partition is considered in turn and the set of simple disjunctive decompositions is found. Partitions where A contains a single variable are ignored as they represent trivial decompositions equivalent to applying Shannon's expansion theorem [1Y] .

Curtis has presented an elaborate theory for determining complex disjunctive decompositions from the set of simple decompositions. A complex disjunctive decomposition is of the form

$$f(A) = g(\alpha_1(A_{\lambda_1}),\alpha_2(A_{\lambda_2}),\ldots,\alpha_t(A_{\lambda_t}),A_\mu)$$

where $A = A_{\lambda_1} \cup A_{\lambda_2} \cup \ldots \cup A_{\lambda_t} \cup A_{\mu_1}$ and A_{λ_1}, A_{λ_2}, ..., A_{λ_t} , and A_μ are mutually disjoint.

Initially, the A_{λ_1}, A_{λ_2}, ..., A_{λ_k} associated with the decompositions are examined for the maximal sets. A set is maximal if it is not a subset of any other set. The decomposition is typed according to whether the maximal sets are mutually conjoint or mutually disjoint, these being the only possibilities. The functions $\alpha_1,\alpha_2,\ldots,\alpha_t$ are determined from their respective decomposition charts, and the function g is determined from these and from the type of the decomposition.

Curtis has also introduced a theory of complex nondisjunctive decomposition. In this case, the A_{λ_i} are not necessarily mutually disjoint. These methods are an extension of the disjunctive case, but the computation is far more involved and extremely lengthy.

The problem of partially specified functions has been treated by Karp [13], Kjelkerud [15], and Hight [10]. In addition, Karp has looked at multi-valued logic, and the multiple-output problem.

A second independent theory of decomposition has been developed by Roth et al [35], [33], [34], and [14]. The cubical calculus is used to represent functions,and 'don't-care' conditions are easily handled. These methods have proven much more general and are more easily programmable than those above.

Because of its more practical applications, we have chosen to introduce decomposition in terms of the second theory. These results will form the basis of the algorithm developed in Chapter 6 . For that reason, emphasis has been placed on the material required for that development. Several interesting results have been excluded or only briefly mentioned for the sake of keeping the fundamentals of the decomposition theory as clear as possible.

2. Boolean Functions and their Representations

Consider a mapping $f : E \to V^1$, where $E \subseteq V^n$, and V^n is the set of all n - tuples of 0's and 1's . Each $e_i \in E$ corresponds to a unique configuration of the Boolean variables $a_1,a_2,\ldots,a_n$, and the mapping $f : E \to V^1$ thus determines a unique Boolean function $f(a_1,a_2,\ldots,a_n)$. If $E = V^n$, $f(a_1,a_2,\ldots,a_n)$ is a total Boolean function. Otherwise, $f(a_1,a_2,\ldots,a_n)$ is a partial function, and the set $V^n - E$ constitutes the 'don't-care' conditions.

In Chapter 3, the cubical complex representation of a total Boolean function was presented. The totality was implicitly assumed as the function was interpreted as false for any vertex not covered by the

complex. We now wish to extend this notation to partial functions.

A partial function may be specified by giving a list of minterms, for which the function value is true, and a list of minterms for which the function value is false. The minterms which do not appear in either list are the 'don't-care' conditions. The lists of vertices corresponding to these minterm lists uniquely define cubical complexes which may be readily found using the cubing algorithm in Chapter 3. Covers of these complexes may also be obtained. The representations of these covers as n - tuples of 0's, 1's, and x's , are termed the ON and OFF arrays, and are denoted C_1 and C_0 .

Definition - A Boolean function $f(a_1,a_2,\ldots,a_n)$ is degenerate if there exists a function

$$g(a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_n) = f(a_1,a_2,\ldots,a_n)$$

for some i , $1 \leq i \leq n$.

The variable a_i is termed 'redundant'. A simple test for redundancy is to form C_1' and C_0' equal to C_1 and C_0 with the i^{th} column removed. If $C_1' \cap C_0' = \phi$, then a_i is redundant. A function with no redundant variables is termed 'non-degenerate'.

3. Decomposition

3.1 Abstract Decomposition

Let X , Y , Z , and W be arbitrary finite sets, and let E be a subset of the Cartesian product $X \times Y$. Given a mapping $f : E \to Z$ the following questions arise:

i) Given $\alpha : X \to W$ does there exist a function $g : W \times Y \to Z$, such that

$$f(x,y) = g(\alpha(x),y) \quad \text{for all} \quad (x,y) \in E$$

ii) Under what conditions do there exist $\alpha : X \to W$, and $g : W \times Y \to Z$ such that

$$f(x,y) = g(\alpha(x),y) \quad \text{for all} \quad (x,y) \in E$$

Such a representation is called a decomposition of f, and g is called the image of the decomposition.

The answers to these questions are based on a compatibility relation between the elements of X.

Definition - $x_i, x_j \in X$ are compatible with respect to f (denoted $x_i \sim x_j$) if, for all $y \in Y$ such that $(x_i,y), (x_j,y) \in E$, $f(x_i,y) = f(x_j,y)$; otherwise, x_i is incompatible with x_j (denoted $x_i \not\sim x_j$).

If $f(a_1,a_2,\ldots,a_n)$ is a total function, then compatibility is an equivalence relation having the properties:

i) $a \sim a$ reflexive

ii) $a \sim b \Leftrightarrow b \sim a$ symmetric

iii) $a \sim b, b \sim c \Rightarrow a \sim c$ transitive

However, if $f(a_1,a_2,\ldots,a_n)$ is a partial function, then compatibility is a quasi-ordered relation and thus does not have the transitive property.

Proofs of the following propositions were presented in [34].

Proposition 5.1

Given f and α, as defined above, there exists g such that

$$f(x,y) = g(\alpha(x),y) \quad \text{for all} \quad (x,y) \in E$$

if, and only if, for all $x_i, x_j \in X$,

$$\alpha(x_i) = \alpha(x_j) \Rightarrow x_i \sim x_j$$

or equivalently

$$x_i \not\sim x_j \Rightarrow \alpha(x_i) \neq \alpha(x_j)$$

Proposition 5.2

If k is the least integer such that X may be partitioned into k classes of mutually compatible elements, then there exist α and g such that

$$f(x,y) = g(\alpha(x),y) \quad \text{for all} \quad (x,y) \in E$$

if, and only if, W has at least k elements.

These propositions establish necessary and sufficient conditions for the existence of a decomposition, and in particular, a decomposition with a predetermined α.

3.2 Decomposition of Boolean Functions

The special case where $f(a_1,a_2,\ldots,a_n)$ is a Boolean function is such that $X \subseteq V^{\ell}$, $Y \subseteq V^{m}$, $W \subseteq V^{t}$, $Z \subseteq V^{1}$, and $\boldsymbol{\alpha}$ represents the t - tuple of Boolean functions $(\alpha_1,\alpha_2,\ldots\alpha_t)$. It is sufficient to consider decompositions of the form

$$f(A) = g(\alpha_1(A_\lambda),\alpha_2(A_\lambda),\ldots,\alpha_t(A_\lambda),A_\mu)$$

where $A = \{a_1, a_2, \ldots, a_n\}$, and $A = A_\lambda \cup A_\mu$, as Hu [11] has shown that any decomposition of a Boolean function can be composed of a finite number of decompositions of this type.

If $A_\lambda \cap A_\mu = \phi$ the decomposition is disjunctive. However, in $A_\lambda \cap A_\mu \neq \phi$, the decomposition is non-disjunctive, and a minor problem arises. Certain variables are common to A_λ and A_μ , and $\ell + m > n$. The domain E of f must be a subset of V^n as f has n - input variables. Therefore, $E \not\subseteq X \times Y$ as $X \subseteq V^\ell$, $Y \subseteq V^m$, and $\ell + m > n$.

The problem is easily solved by defining $\delta : V^n \to V^{\ell+m}$ such that a cube $v = \{v_1, v_2, \ldots, v_n\}$ has the image $\{v_1, v_2, \ldots, v_n, v_{i_1}, v_{i_2}, \ldots, v_{i_p}\}$ $p = \ell + m - n$, where $A_\lambda \cap A_\mu = \{a_{i_1}, a_{i_2}, \ldots, a_{i_p}\}$. We can then deal with the function $\hat{f}(A_\lambda, A_\mu)$ which is such that if $\delta : v \to \hat{v}$, then $\hat{f}(\hat{v}) = f(v)$.

Let us suppose that a given function $f(A)$ is represented by covers $C_1 = \{b_i,\ i = 1,2,\ldots,n\}$, and $C_0 = \{c_j,\ j = 1,2,\ldots,m\}$. We wish to employ propositions 5.1 and 5.2 for the detection of a decomposition relative to a given $f(A)$, and partition of A . Any cube $b_i \in C_1$ or C_0 can be divided into a 'λ part', b_{i_λ} , and a 'μ part', b_{i_μ} . The co-ordinates of b_{i_λ} are the co-ordinates of b_i corresponding to variables in A_λ , and the co-ordinates of b_{i_μ} are the co-ordinates of b_i corresponding to variables in A_μ . The covers C_1 and C_0 are thus represented as

$$C_1 = \{(b_{i_\lambda}, b_{i_\mu}),\ i = 1,2,\ldots,n\}$$

$$C_0 = \{(c_{j_\lambda}, c_{j_\mu}),\ j = 1,2,\ldots,m\} \quad .$$

The following lemma [34] is a method for determining the compatibility relations of the elements of X relative to a given function specified by C_1 and C_0.

Lemma 5.1

Given $v_0 \varepsilon V^{\ell}$ and $v_1 \varepsilon V^{\ell}$, $v_0 \not\gamma v_1$ if, and only if, there are cubes $(b_{i_\lambda}, b_{i_\mu}) \varepsilon C_1$ and $(c_{j\lambda}, c_{j_\mu}) \varepsilon C_0$ such that

i) $b_{i_\mu} \cap c_{j_\mu} \neq \phi$

ii) either $b_{i_\lambda} \supseteq v_0$ and $c_{j_\lambda} \supseteq v_1$, or

$c_{j_\lambda} \supseteq v_0$ and $v_{i_\lambda} \supseteq v_1$.

For example, consider $f(a,b,c,d)$ given by the covers C_1 and C_0 below. We wish to

C_1

	a	b	c	d
b_1	1	0	1	x
b_2	1	x	1	0
b_3	0	1	x	x

C_0

	a	b	c	d
c_1	1	1	x	1
c_2	1	x	0	x
c_3	x	0	0	x

determine if a decomposition exists for $A_\lambda = \{a,b\}$, and $A_\mu = \{c,d\}$.

Consider the cubes b_1 and c_1. Here, $b_{1_\lambda} = 10$, $b_{1_\mu} = 1x$, $c_{1_\lambda} = 11$, and $c_{1_\mu} = x1$. $b_{1_\mu} \cap c_{1_\mu} = 1x \cap x1 = 11$ which is not ϕ. Therefore by lemma 5.1, $b_{1_\lambda} \not\gamma c_{1_\lambda}$. Examining all pairs of cubes, one from each cover, we find all the incompatibilities. They are

$10 \not\sim 11$

$01 \not\sim 11$

$01 \not\sim 1x$ (i.e. $01 \not\sim$ any vertex covered by the cube $1x$)

$01 \not\sim x0$

Propositions 5.1 and 5.2 can now be used to determine the nature of the possible decompositions.

Suppose $\alpha(a,b) = a \cdot b$. We wish to determine if g exists such that

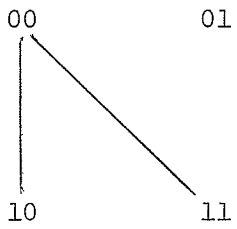
$$f(a,b,c,d) = g(\alpha(a,b),c,d) \quad .$$

Since $\overline{a} \cdot b = a \cdot \overline{b}$, but $01 \not\sim 10$, proposition 5.1 indicates that no suitable g , and α , exist.

As a more general problem let us determine if any g and α exist such that

$$f(a,b,c,d) = g(\alpha(a,b),c,d) \quad .$$

Since $\alpha(a,b)$ is Boolean, its range is V^1 which has two members. According to proposition 5.2 a decomposition exists if, and only if, X , the domain of α , can be partitioned into two sets of mutually compatible elements. The following diagram shows the elements of X with lines connecting compatible pairs.



It is clear that a suitable partition cannot be found, and therefore no decomposition exists.

Lemma 5.1 only applies in the disjunctive case. In the non-disjunctive case we must again deal with $\hat{f}$ which implies that we must find $\hat{C}_1$ and $\hat{C}_0$ from C_1 and C_0. The obvious solution is to simply replace each cube b_i of C_0 and C_1 by a cube with the co-ordinates common to b_{i_λ} and b_{i_μ} appearing twice. For example, if $A_\lambda = \{a,b,c\}$ and $A_\mu = \{c,d\}$, the cube

a	b	c	d
x	0	1	x

would be replaced by

A_λ			A_μ	
a	b	c	c	d
x	0	1	1	x

Unfortunately, inconsistencies can occur. The cube

a	b	c	d
1	0	x	x

would be replaced by

a	b	c	c	d
1	0	x	x	x

.

This cube contains the cubes 1000x, 1011x, 1001x, and 1010x. The latter two cubes are inconsistent as the variable c simultaneously takes on the values 1 and 0. A cube with q x's in co-ordinates corresponding to the variables of $\{A_\lambda \cap A_\mu\}$ must be replaced by 2^q cubes representing the consistent assignments.

In practice, $\hat{C}_1$ and $\hat{C}_0$ are not actually formed, but certain columns of C_1 and C_0 are thought of as appearing both in the λ and μ parts of each cube.

Consider the example above with $A_\lambda = \{a,b\}$, and $A_\mu = \{b,c,d\}$. b_{1_μ} no longer intersects c_{1_μ} as $01x \cap 1x1 = \phi$. The following incompatibilities are found

$i = 3$, $j = 1$ $\quad 01 \not\sim 11$

$i = 3$, $j = 2$ $\quad 01 \not\sim 11$ (N.B. $01 \sim 10$)

Since $00 \sim 01$, $00 \sim 10$, and $01 \sim 10$, f can be rewritten

$$f(a,b,c,d) = g(a \cdot b,\ b,c,d) \quad .$$

An alternative approach to the non-disjunctive case is to treat the variables of $\{A_\lambda \cap A_\mu\}$ as if they were Boolean functions of one variable produced by some $\alpha_i \in (\alpha_1,\alpha_2,\ldots,\alpha_t)$. This reduces the non-disjunctive case to the disjunctive case.

When $A_\lambda = \{a,b\}$, and $A_\mu = \{c,d\}$, we had the incompatibilities

$$10 \not\sim 11$$

$$01 \not\sim 1x$$

$$01 \not\sim x0$$

Suppose $\alpha_1(a,b) = a \cdot b$, and $\alpha_2(a,b) = b$. By inspecting each incompatibility we find that proposition 5.1 is not violated, and the decomposition

$$f(a,b,c,d) = g(\alpha_1(a,b),\alpha_2(a,b),c,d)$$

is valid. This approach will be considered in detail in Chapter 6.

Lemma 5.1 can be applied in two ways:

i) In conjunction with proposition 5.1 , it can be used to determine, given, f , A_λ , A_μ , and $(\alpha_1,\alpha_2,\ldots,\alpha_t)$ whether g exists such that

$$f(A) = g(\alpha_1(A_\lambda),\alpha_2(A_\lambda),\ldots,\alpha_t(A_\lambda),A_\mu)$$

or ii) In conjunction with proposition 5.2 , it can be used to determine, given f , A_λ , and A_μ , whether g , and $(\alpha_1,\alpha_2,\ldots,\alpha_t)$ exist such that

$$f(A) = g(\alpha_1(A_\lambda),\alpha_2(A_\mu),\ldots,\alpha_t(A_\lambda),A_\mu) \quad .$$

The first case is straightforward. The incompatibilities of elements of X with respect to f are found, and each is checked for a violation of proposition 5.1 . If no violation is found the decomposition is valid.

The second case is more complex. Let k be the least integer such that X , the domain of α , may be partitioned into k classes of mutually compatible elements. From proposition 5.2 , if a decomposition exists, then W , the range of α , must have at least k elements. As α represents the t - tuple $(\alpha_1,\alpha_2,\ldots,\alpha_t)$ of Boolean functions, $k \leq 2^t$ is a necessary and sufficient condition for the existence of a decomposition. The determination of k is thus extremely important.

Definition - $S \subseteq X$ is a compatible set if $x_i , x_j \in X \Rightarrow x_i \sim x_j$.

Definition - S is a maximal compatible set if for any T such that $S \subset T \subseteq X$, T is not a compatible set.

The first step in determining k is to find the maximal compatible sets of X . Roth and Karp [34] have presented an algorithm based on the state reduction techniques for sequential machines due to R. E. Miller [22] . Once these sets are found the general extraction algorithm of Roth and Wagner [35] is used to find the minimum number of maximal compatible sets whose union is X . These sets form a cover

of X which is easily transformed into a partition of X by the removal of elements common to other sets.

Roth and Karp have also introduced a simplified algorithm for the case of $t = 1$. This is of interest as the majority of practical switching elements realize a single output function.

A second problem of interest is the case where α is a vertex function. A vertex function is such that $\alpha(v_0) \neq \alpha(v)$, where v is any other vertex. v_0 is termed the distinguished vertex. The AND, OR, NAND, and NOR are vertex functions; the EXCLUSIVE OR and MAJORITY are not. Primitive devices constructed of diodes, vacuum tubes, or transistors normally realize a vertex function. Special techniques for vertex decomposition have been developed by Roth and Karp, and Barnard and Holman [2].

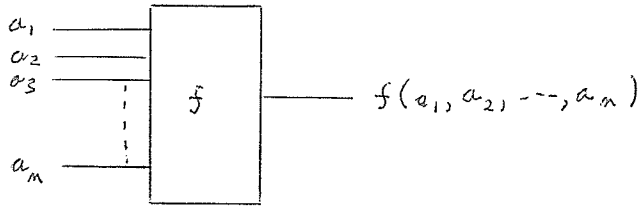
In circuit design, we shall be concerned with sequences of decompositions, each operating on the image of its predecessor. The computation of these images is thus an important link in the design process. Roth and Karp [34] have presented an algorithm for finding the image of a decomposition of the form

$$f(A) = g(\alpha_1(A_\lambda), \alpha_2(A_\lambda), \ldots, \alpha_t(A_\lambda), A_\mu) \quad .$$

In Chapter 6 a table lookup procedure is developed for the case where the α_i are two-place functions. The work involved in computing the image is minimal and the covers determining the image are quite compact.

4. Circuit Design Algorithms

A switching circuit is often represented as a black box with only the inputs and outputs specified.

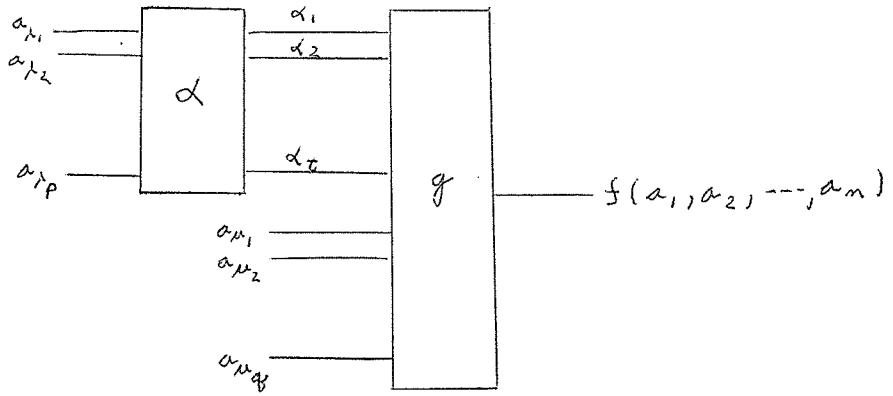


The problem is to construct the interior of the black box by interconnecting primitive switching elements of given types.

The decomposition

$$f(A) = g(\alpha_1(A_\lambda), \alpha_2(A_\lambda), \ldots, \alpha_t(A_\lambda), A_\mu)$$

represents the circuit



Decompositions are chosen so that the α_i represent allowed switching elements, and a circuit is found by successively decomposing each image until an image representing an allowed switching element is found. A sequence of decompositions of this form is termed a total decomposition.

Karp, McFarlin, Roth, and Wilts [14] have presented an algorithm for the design of single-output circuits constructed from an

arbitrary set of primitive logic elements. The cost of the circuit, taken to be the sum of the costs of the individual switching elements, is minimized. An exhaustive search is made of all sequences of decompositions whose costs are less than a prescribed bound. This bound shrinks as more economical circuits are found, and the search terminates when all circuits below the lowest cost bound are accounted for.

Cycling is a phenomenon where a new variable is equal to:

i) a constant

ii) a primary input or its complement

iii) a variable produced by a previous decomposition or the complement of such a variable.

Cycling never results in minimal circuits and steps were taken to avoid it.

Karp et al. found that an exhaustive search was not practical for problems of more than three of four variables. Methods were therefore developed to make the initial circuit as minimal as possible. At each stage, as might be expected, the decomposition corresponding to the least expensive building block is chosen. Also, the 'd - algorithm' of Ashenhurst [1] is used to trace more than one sequence of decompositions at a time. The choice of the best sequence is thus delayed until more information is known. The circuits in figures 5.1 and 5.2 and 5.3 were designed using this program.

Figure 5.1 is the two out of five checker presented in Chapter 4. This circuit was originally given in terms of AND, OR, and NOT gates. We have substituted NAND gates to facilitate its comparison to the results of other methods. Clearly, this result is much better than the circuit found by factoring.

FIGURE 5.1

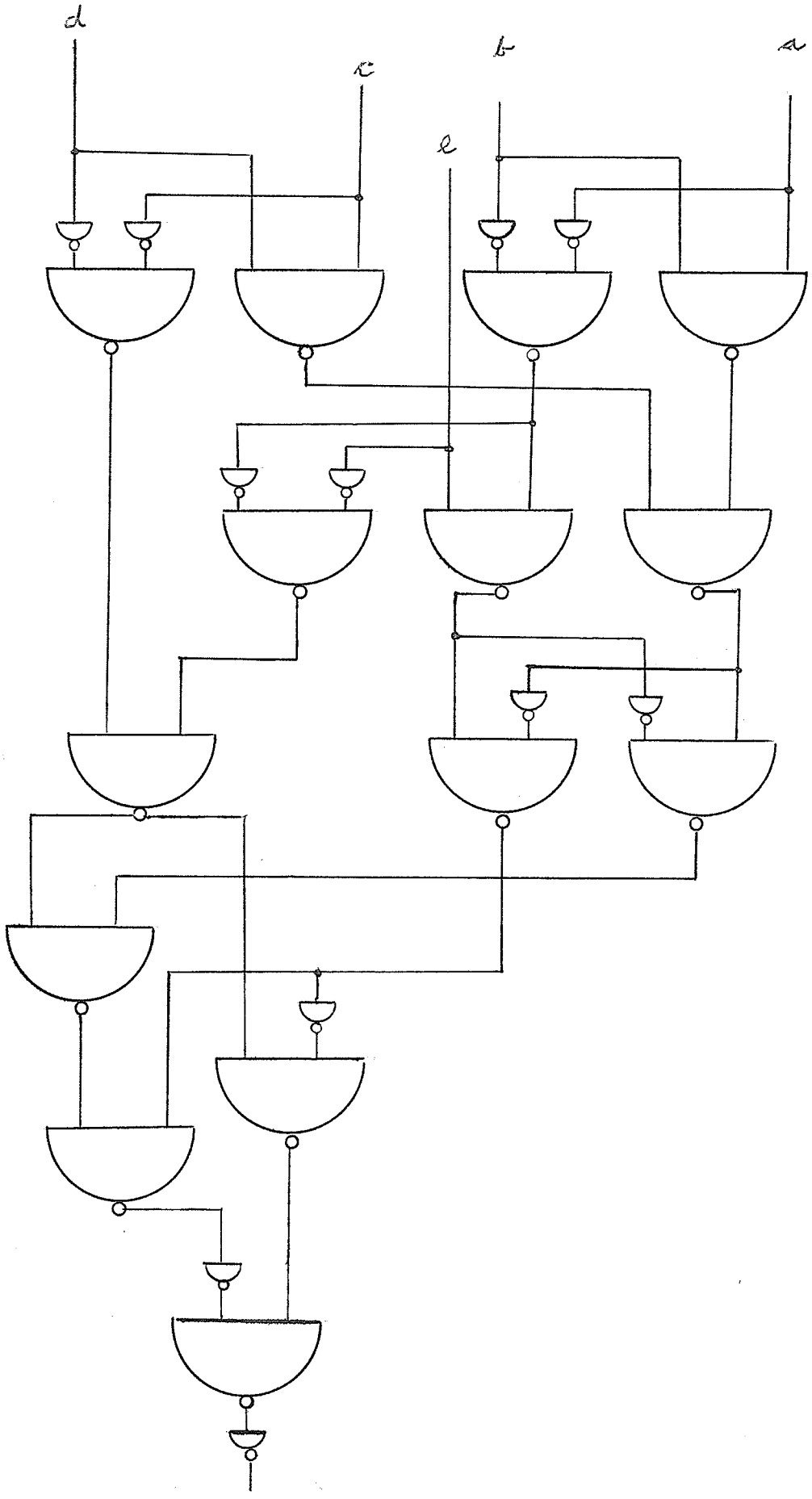


Figure 5.3 is the initial program design for a function without the use of the 'd - algorithm' . Figure 5.2 is the same function, but the 'd - algorithm' was used. The improvement is obvious. In these diagrams we have presented the dual of the actual problem solved. This is again to facilitate comparison to other methods.

Barnard and Holman [2] have presented an algorithm similar to that above. A rapid technique for determining two-place vertex decompositions has been developed, but any advantage gained was lost by not attempting to optimize the initial circuit. For example, figure 5.4 illustrates their initial approximation to the function of figures 5.2 and 5.3 . A result nearly identical to figure 5.2 was found after considerable searching which required over a minute of CPU time on a KDF9 computer.

In Chapter 6, an algorithm is developed which combines the good points of each of the previous decomposition algorithms, while avoiding their obvious shortcomings. Emphasis is placed on the speed of computation, and on obtaining a nearly minimal result in one pass. The algorithm is restricted to two-input NAND gates, but as shown in Chapter 7, it is easily extended to multiple-input gates of any type.

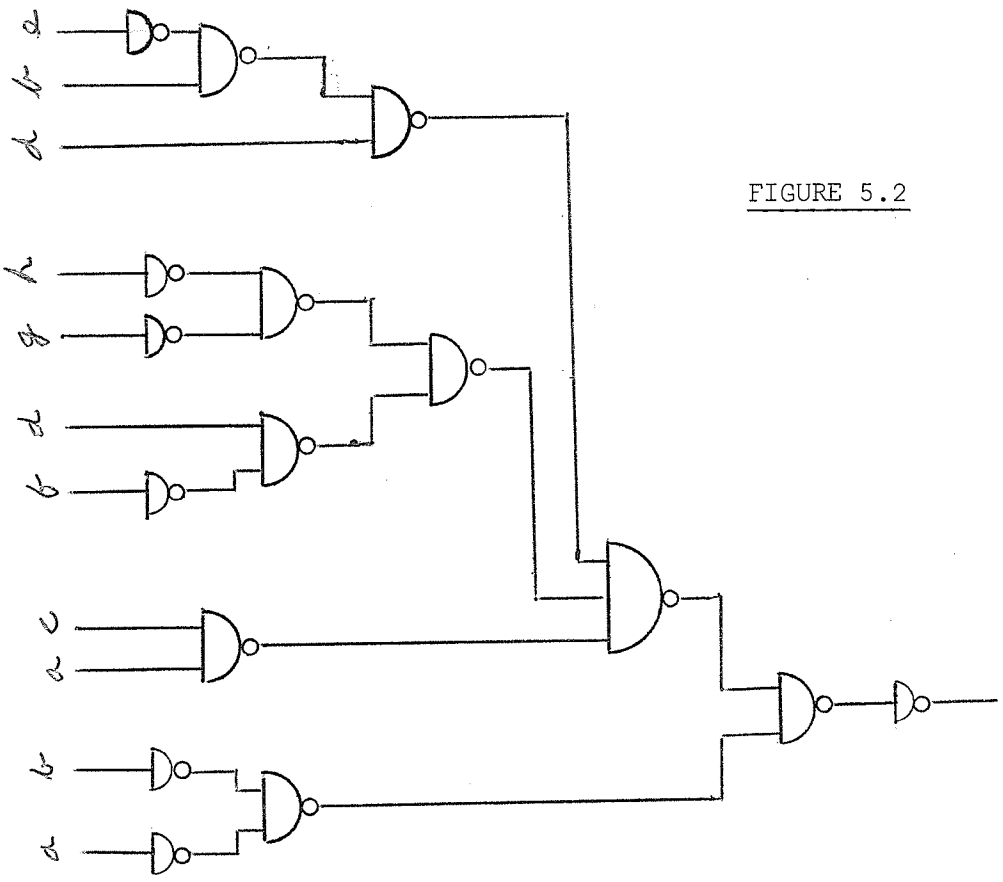


FIGURE 5.2

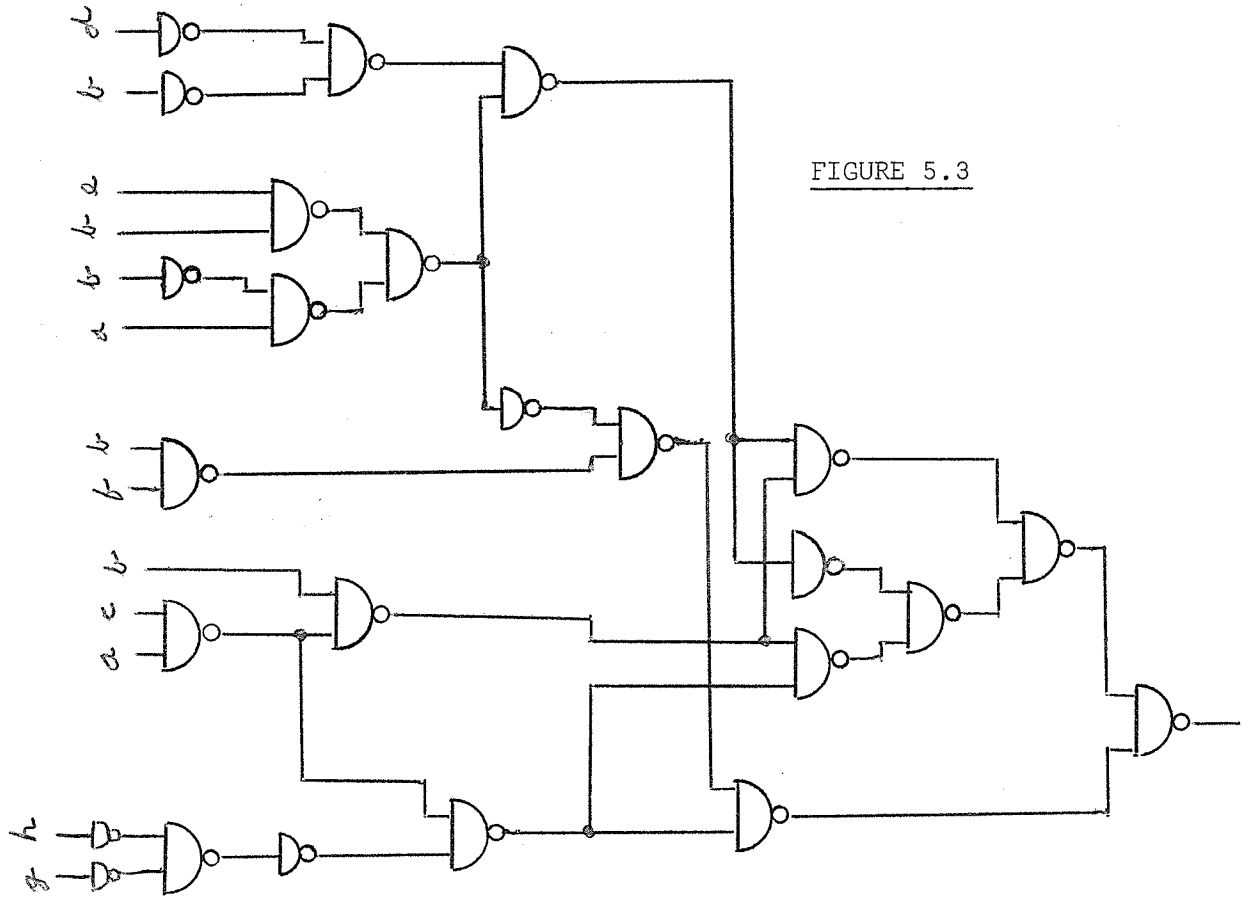
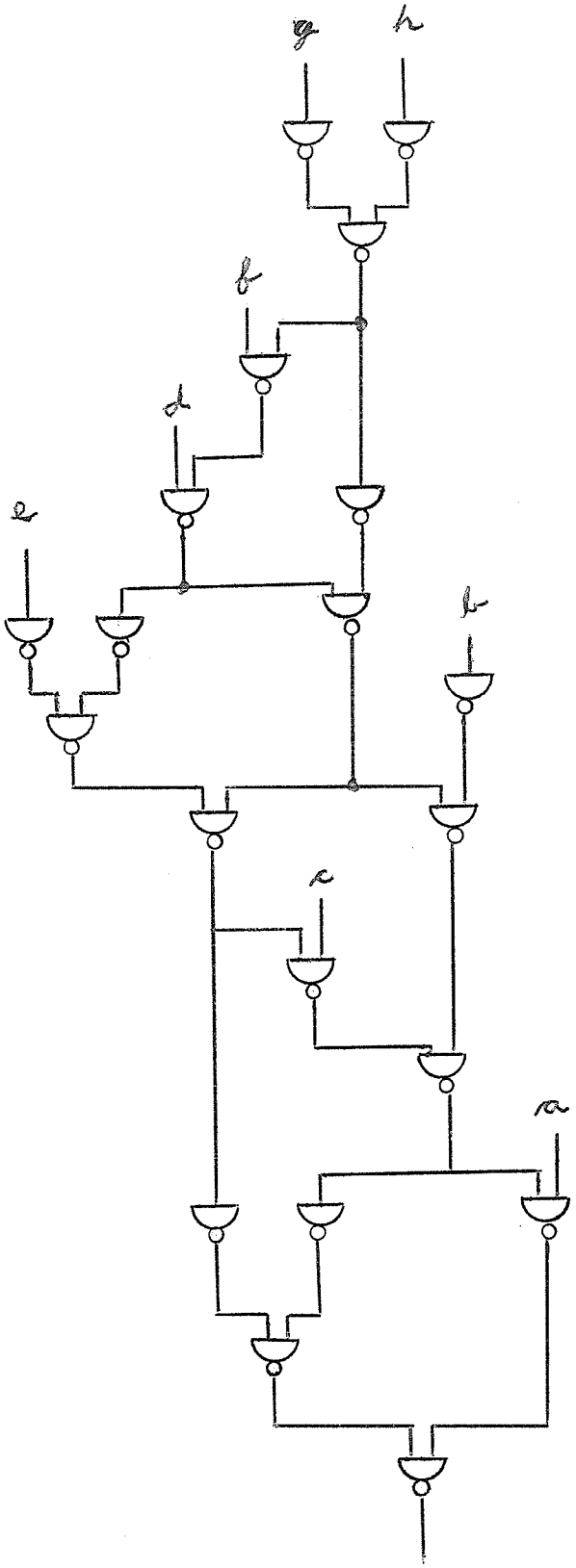


FIGURE 5.3

FIGURE 5.4



Chapter 6

The Design of NAND Circuits by Decomposition

1. Introduction

In this chapter, the total decomposition of a partial Boolean function by successive two-place decompositions is discussed. A theory of two-place decomposition is developed, and algorithms for its efficient implementation are presented. These methods form the basis of an heuristic total decomposition algorithm specifically designed to yield an expression corresponding to a NAND circuit of nearly minimal cost.

This algorithm has been used in a computer program for the design of circuits constructed of one and two-input NAND gates. The program is briefly described with particular emphasis placed upon the techniques used to eliminate redundant computation.

2. Two-Place Decomposition Theory

Let F be a many to one mapping of E onto Z, and let E be a subset of V^n and Z be a subset of V^1, where V^m is the set of all m-tuples (ordered sets of length m) of zeros and ones. There exist sets X and Y such that X is a subset of V^2, Y is a subset of V^{n-2}, and E is a subset of the Cartesian product $X \times Y$. If there exists a mapping $\alpha: X \to W$, where W is a subset of V^2, and a mapping $G: W \times Y \to Z$ such that, for all $(x,y) \in E$, $F(x,y) = G(\alpha(x),y)$, then F is said to be two-place decomposable.

Corresponding to F and G are unique Boolean functions $f(a_1,a_2,\ldots,a_n)$ and $g(b_1,b_2,\ldots,b_m)$. The mapping α corresponds to

the two-tuple of Boolean functions $(\alpha_1(c_1,c_2), \alpha_2(c_1,c_2))$, and the decomposition above may be written

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_1(a_1,a_2), \alpha_2(a_1,a_2),a_3,\ldots,a_n) \quad .$$

The term two-place decomposition is used since $\alpha_1(a_1,a_2)$ and $\alpha_2(a_1,a_2)$ are two-place functions.

Two-place decompositions are typed according to the degeneracy of α_1 and α_2 . The resulting types are:

i) degenerate: Neither α_1 or α_2 is a true two-place function. Conditions will be found for which variables are redundant in $f(a_1,a_2,\ldots,a_n)$.

ii) simple disjunctive: One of α_1 and α_2 is a constant and the other a true two-place function. Decompositions of this type may be rewritten in the form

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_i(a_1,a_2),a_3,\ldots,a_n)$$

where $i \in \{1,2\}$.

iii) simple non-disjunctive: One of α_1 and α_2 is a true two-place function and the other a one-place function. The decomposition may in this case be rewritten as

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_i(a_1,a_2), a_j,a_3,\ldots,a_n)$$

where $i, j \in \{1,2\}$.

iv) complex disjunctive: Both α_1 and α_2 are true two-place functions and the form of the decomposition cannot be simplified.

Using the cubical calculus, a method for determining the decomposition relative to a given function and two-place partition will be

developed. The problem of finding the simplest equivalent decomposition will also be solved.

3. The Determination of Two-Place Decompositions

From proposition 5.2 a necessary and sufficient condition for the existence of α and G is that W contain at least k elements, where X may be partitioned into k classes of mutually compatible elements.

In the general case, where $f(a_1,a_2,\ldots,a_n)$ is a partial function, k is the minimum number of disjoint compatible subsets of X which form a cover of X. Roth's method [34] for determining k is based on an algorithm due to R. E. Miller [22] for the reduction of states in sequential machines. Several such algorithms exist and can be adapted to the problem at hand, but they are general in nature and a more efficient method based on the specific properties of this problem will be developed.

3.1 Compatibility Tables and Compatible Sets

The compatibility relation between the elements of X with respect to a given F was defined in the previous chapter. The following definitions are also required.

Definition - A compatible set, $S \subseteq X$, is such that for all $x_i, x_j \in S$, $x_i \sim x_j$.

Definition - A compatible set, $S \subseteq X$, is termed maximal if, and only if, for all T, $S \subset T \subseteq X$, T is not a compatible set. Maximal compatible sets will be denoted $M_1,M_2,\ldots,M_i,\ldots$ and the set of all such sets M.

Definition - $M' \subseteq M$ is a minimum cover of X if, and only if, each $x_i \in X$ is in at least one $M_j \in M'$ and there exists no cover with fewer sets.

Definition - A compatibility table T whose i^{th} row and column are labelled x_i is a set of n-tuple representation of the sets $S_1, S_2, \ldots, S_n$. Each S_i is such that $x_j \in S_i$ if, and only if, $x_i \sim x_j$ and $x_j \notin S_i$ if, and only if, $x_i \not\sim x_j$. A compatibility table is symmetric and has a unit diagonal as compatibility is a symmetric and reflexive relation.

Definition - The lower bound L of a set of sets R is such that for all $R_i, R_j \in L$, $R_i \not\subseteq R_j$, $R_j \not\subseteq R_i$, and $R_i, R_j \in R$.

We will denote the lower bound of the sets $S_1, S_2, \ldots, S_n$ as L. The use of L in the determination of k is demonstrated by the following results.

Theorem 6.1

Every maximal compatible set in M' is a subset of some $S_K \in L$.

Proof

Suppose $L = \{S_{i_1}, S_{i_2}, \ldots, S_{i_n}\}$ and M' contains an M_j such that $x_{i_s} \notin M_j$ for any s, $1 \subseteq s \subseteq n$. Let $M_j = \{x_{j_1}, x_{j_2}, \ldots, x_{j_m}\}$ where $j_n \neq i_s$, $1 \leq r \leq m$, $1 \leq s \leq n$. For each r, s_{i_r} must have been subsumed by some s_{i_s}, so x_{i_r} is compatible with all its elements and must be included as M_i is maximal compatible. Thus, M_j is covered

by other sets of M' and is redundant. M' is therefore non-minimal. We conclude that if M' is a minimal cover, then $M_i \subseteq L$ for all $M_i \in M'$.

It follows immediately that

Corollary 6.1.1

Any maximal compatible set which is deducible only from some $s_j \notin L$ is redundant in a minimal cover.

Theorem 6.2

If each $S_i \in L$ is a maximal compatible set, then L is an irredundant cover of X.

Proof

Consider $x_i \in S_i \in L$. If $x_i \in S_i$ then $x_i \sim x_j \Leftrightarrow x_i \in S_j$, and if $x_j \notin S_i$, then $x_j \not\sim x_i \Leftrightarrow x_i \notin S_j$. Initially we show that x_i is a distinguished element if S_i is a maximal compatible set, that is $x_i \notin S_j$, any $S_j \in L$, $i \neq j$. To prove x_i is distinguished it is sufficient to show there is no $S_j \in L$ such that $x_j \in S_i$.

Consider S_j such that $x_j \in S_i$. Clearly, $S_j \neq S_i$ since $S_i \in L$. S_i, S_j are partially disjoint (they cannot be disjoint since $x_i, x_j \in S_i$ and $x_i, x_j \in S_j$). There exists $x_k \in S_i$, $x_k \notin S_j$ so $x_k \not\sim x_j$. But $x_j \in S_i$ and S_i is maximal compatible set so $x_k \sim x_j$ since $x_j \in S_i$. This is a contradiction and it follows that x_i is distinguished.

Now each maximal compatible set contains a distinguished element and trivially no set can contain two such vertices since

$x_i , x_j \in S_i \Rightarrow x_i , x_j \in S_j$ and if $S_i , S_j \in L$, then x_i, x_j are not distinguished. Therefore, each $S_i \in L$ is essential and L is an irredundant cover.

Corollary 6.2.1

The irredundant cover of theorem 6.2 is also minimal. This follows immediately from theorems 6.1 and 6.2 .

Criterion 6.1

$S_i \in L$ is not a maximal compatible set if, and only if, there exist $x_j , x_k \in S_i$ $(i \neq j, i \neq k, j \neq k)$ and two other sets $S_p , S_q \in L$ such that

i) $x_j \in S_p , x_j \notin S_q$

ii) $x_k \in S_q , x_k \notin S_p$

and iii) $T_{j_k} = 0$ are all satisfied.

(T is the original compatibility table.)

Proof

Clearly, if the conditions hold, S_i is not a maximal compatible set since $T_{j_k} = 0$ implies $x_j \not\sim x_k$.

Conversely, suppose S_i is not a maximal compatible set; then either

i) S_i is a compatible set but not maximal;

ii) S_i is not a compatible set.

We consider the two possibilities in turn.

i) If S_i is a compatible set but is not maximal, there exists

an M_j such that

$$S_i \subset M_j \subseteq X \quad .$$

There is an $x_k \in M_j$, $x_k \notin S_i$. But $x_i \in M_j$ implies $x_i \sim x_k$ and $x_k \notin S_i$ implies $x_i \not\sim x_k$. Therefore if S_i is a compatible set, it is also maximal.

ii) implies there exist x_j , $x_k \in S_i$ such that $x_j \not\sim x_k$.

Hence, $T_{j_k} = 0$, and $x_j \in S_k$, $x_k \notin S_j$. Now either $S \in L$ or there is an $S_r \in L$ such that $S_r \subset S_k$. If $S_r \subset S_k$ then $x_r \in S_k$ so $T_{r_k} = 1 = T_{k_r}$ and $x_k \in S_r$. Further, $x_j \notin S_r$. Similarily, either $S_j \in L$ or there is an $S_r \in L$ such that $S_s \subset S_j$, $x_j \in S_s$, $x_k \notin S_s$.

This gives the conditions of the criterion.

A general discussion of compatability and compatible sets, including the above material is given in [20] .

The problem of present interest is the determination of k when X is a subset of V^2 . A stronger criterion for the equality of L and M' under this condition will be established.

Definition - A Boolean function $f(a_1,a_2,\ldots,a_n)$ is degenerate if there exists $g(b_1,b_2,\ldots,b_m) = f(a_1,a_2,\ldots,a_n)$ where $m < n$, $b_i \in a_1,a_2,\ldots,a_n$, and $b_i \neq b_j$, $1 \leq i \leq m$, $1 \leq j \leq m$.

We shall denote the q^{th} co-ordinate of $x_i \in X$ as x_{i_q} .

Theorem 6.3

If $x_i \sim x_j$ and $x_k \sim x_\ell$, x_i , x_j , x_k , $x_\ell \in X \subseteq V^2$, and $x_{i_q} = x_{j_q}$, then $f(a_1,a_2,\ldots,a_n)$ is degenerate.

Proof

$x_{i_q} = x_{j_q} \Rightarrow x_{k_q} = x_{\ell_q}$ as $X \subseteq V^2$. Let α be such that $\alpha(x_i) = \alpha(x_j) = x_{i_q}$, and $\alpha(x_k) = \alpha(x_\ell) = x_{k_q}$. Corresponding to α is a Boolean function $\alpha_1(a_1,a_2) = a_q$ and by proposition 5.2

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_1(a_1,a_2),a_3,\ldots,a_n)$$

$$= g(a_q,a_3,\ldots,a_n)$$

$f(a_1,a_2,\ldots,a_n)$ is thus a degenerate function.

Theorem 6.4

If $X \subseteq V^2$, and $S_i \in L$ is not maximal compatible, then $f(a_1,a_2,\ldots,a_n)$ is a degenerate function.

Proof

From criterion 6.1 there exist x_j , $x_k \in S_i$ $(i \neq j, i \neq k, j \neq k)$ and S_p , $S_q \in L$ such that

i) $x_i \in S_p$, $x_j \notin S_q$

ii) $x_k \in S_q$, $x_k \notin S_p$

iii) $T_{j_k} = 0$

$$S_i , S_p \in L \Rightarrow x_a \in S_p , x_a \notin S_i .$$

$$S_i , S_q \in L \Rightarrow x_b \in S_q , x_b \notin S_i .$$

$x_i , x_j,, x_k , x_a$, and $x_b \in X$. But, $X \subseteq V^2$ and i, j, and k are distinct. Therefore $x_a = x_b = x_\ell$. $x_k \notin S_p$ and either $p = \ell$, or $p = j$. Therefore $x_j \sim x_\ell$. $x_j \notin S_q$ and either $q = \ell$, or $q = k$. Therefore $x_k \sim x_\ell$. x_i is adjacent to two elements in X and there thus exists a q such that $x_{i_q} = x_{j_q}$ or $x_{i_q} = x_{k_q}$. Theorem 6.3 is satisfied and $f(a_1,a_2,\ldots,a_n)$ is therefore degenerate.

Criterion 6.2

If $X \subseteq V^2$ and $f(a_1,a_2,\ldots,a_n)$ is non-degenerate, then $M' = L$.

Proof

This follows immediately from theorem 6.4 and corollary 6.2.1 .

The determination of M' when $f(a_1,a_2,\ldots,a_n)$ is non-degenerate, and $X \subseteq V^2$ is straightforward. In general, however, this minimum cover is not necessarily disjoint and therefore does not represent the partition required by proposition 5.2 .

The following method [34] can be used to find a minimum partition M" .

Let M_i (i = 1,...,k) be the maximal compatible sets of M' . Determine the disjoint sets N_i (i = 1,...,k) of M" as follows:

$$N_1 = M_1$$

$$N_i = M_i - (M_i \cap \bigcap_{j=1}^{i-1} M_j) \text{ for each } i = 2,\ldots,k .$$

This method does not, unfortunately, always yield the partition of X which represents the simplest decomposition.

Let $M' = \{M_1 = \{00,11\}, M_2 = \{10,01,11\}\}$. By the above method we obtain $M'' = \{N_1 = \{00,11\}, N_2 = \{10,01\}\}$. The simplest possible assignment of α gives $\alpha_1(a_1,a_2) = 0$, and $\alpha_2(a_1,a_2) = a_1\bar{a}_2 + \bar{a}_1 a_2$. Suppose $M'' = \{N_1 = \{00\}, N_2 = \{10,01,11\}\}$ which is a valid compatible partition. Here α could be assigned so that $\alpha_1(a_1,a_2) = 0$, and $\alpha_2(a_1,a_2) = a_1 + a_2$ which is more desirable. An algorithm has been developed which for $X \subseteq V^2$ yields the partition M'' corresponding to the simplest decomposition. The derivation begins by considering the general case where $X \subseteq V^n$.

Let α be a mapping of X onto W, $\alpha : X \to W$, where $X \subseteq V^n$, and $W \subseteq V^n$. α corresponds to its n-tuple of Boolean functions $\{\alpha_1(a_1,\ldots,a_n), \alpha_2(a_1,\ldots,a_n), \ldots, \alpha_n(a_1,\ldots,a_n)\}$. Let F be a non-degenerate mapping of E onto Z, $F : E \to Z$, where $E \subseteq X \times Y$, $Y \subseteq V^{m-n}$, and $Z \subseteq V^1$. Proposition 5.1 is satisfied as the maximum number of classes of compatible elements in X is the number of elements in V^n. Let M'' be a partition of X such that for all $N_i \in M''$, N_i is a compatible set. We denote the cordinality of a set S as $[S]$.

Theorem 6.5

α may be assigned to satisfy proposition 5.2 such that there are m 0-place functions in $\{\alpha_1(a_1,\ldots,a_n), \ldots, \alpha_n(a_1,\ldots,a_n)\}$ if, and only if, $[M''] \leq 2^{n-m}$.

Proof

Assume $[M''] \leq 2^{n-m}$.

It follows that W contains at most 2^{n-m} elements. Let the k^{th} element of W be the vertex represented by the n-place binary number $k - 1$. Clearly, the first m positions of every element are 0 as the largest number $2^{n-m} - 1$ requires $n - m$ binary digits. The m functions α_i, $i = 1, m$, are thus constant or 0-place functions.

Conversely, assume there are m 0-place functions in $\{\alpha_1(a_1,\ldots,a_n),\ldots,\alpha_n(a_1,\ldots,a_n)\}$ and $[M''] > 2^{n-m}$. m positions of the binary representations of the α_i are constant. There are at most 2^{n-m} distinct elements in W and as $[M''] > 2^{n-m}$ proposition 5.2 cannot be satisfied.

Corollary 6.5.1

If $2^{\ell-1} < [M''] \leq 2^{\ell}$ then the n-tuple $\{\alpha_1(a_1,\ldots,a_n),\ldots,\alpha_n(a_1,\ldots,a_n)\}$ must contain at least ℓ α_i which are not 0-place functions.

Theorem 6.6

α may be assigned so that α_i is a 1-place function if, and only if, for all x_j, $x_k \in S_\ell \in M''$, $x_{j_i} = x_{k_i}$.

Proof

Assume for all x_j, $x_k \in S_\ell \in M''$, $x_{j_i} = x_{k_i}$. Let α be such that for all $S_\ell \in M''$

$$\alpha(x_j) = x_k \; ; \; x_j, x_k \in S_\ell$$

$x_{j_i} = x_{k_i}$ as x_j, $x_k \in S_\ell$, and therefore, $\alpha_i(a_1,a_2,\ldots,a_n) = a_i$, a 1-place function.

Conversely assume $\alpha_i(a_1,\ldots,a_n)$ is a 1-place function and $x_{j_i} \neq x_{k_i}$, x_j, $x_k \in S_\ell$, for some $S_\ell \in M''$.

$\alpha(x_j) = \alpha(x_k) = w_q \in W$ as x_j, $x_k \in S_\ell$. As $\alpha_i(a_1,\ldots,a_n)$ is a 1-place function, $w_{q_i} = x_{j_i}$ and $w_{q_i} = x_{k_i}$ which implies $x_{j_i} = x_{k_i}$. This is a contradiction and $\alpha_i(a_1,\ldots,a_n)$ cannot be a 1-place function.

Conditions have thus been established for when $\alpha_i(a_1,\ldots,a_n) \in \{(\alpha_1(a_1,\ldots,a_n),\ldots,\alpha_n(a_1,\ldots,a_n)\}$ can be assigned as a zero or one-place function.

Definition - If $\alpha_i(a_1,a_2,\ldots,a_n)$ is a true k-place function, the order of $\alpha_i(a_1,a_2,\ldots,a_n)$ is k.

Definition - The order of a decomposition

$$f(a_1,a_2,\ldots,a_n) = q(\alpha_1(a_1,\ldots,a_m),\ldots,\alpha_m(a_1,\ldots,a_m),a_{m+1},\ldots,a_n)$$

is the sum of the orders of the α_i.

We wish to derive M'' from M' so that the subsequent decomposition has minimal order. At present, we are interested in two-place decompositions where $n = 2$. A method for the general case is under consideration.

As $X \subseteq V^2$, the cardinality of M'' is less than or equal to four. Clearly, $[M''] = 1$ cannot occur as this implies a_1 and a_2 are redundant in $f(a_1,a_2,\ldots,a_n)$ which violates the assumption of non-degeneracy. The case where $[M''] = 4$ is a special case where each $S_i \in M''$ contains only $x_i \in X$. x_i agrees with itself in both the first and second co-ordinates and thus, by theorem 6.6 $\alpha_1(a_1,a_2) = a_1$ and $\alpha_2(a_1,a_2) = a_2$.

The decomposition in this case is termed totally degenerate. We shall treat the remaining cases, $[M''] = 2$, and $[M''] = 3$, seperately.

i) $[M''] = 2$

By corollary 6.5.1 and theorem 6.5 , one of α_1 or α_2 can be assigned to be a zero-place function and the other cannot. The one that cannot must be a true two-place function as $f(a_1,\ldots,a_n)$ is non-degenerate. The minimum order of a decomposition where $[M''] = 2$ is therefore also 2 . Such decompositions are simple disjunctive.

ii) $[M''] = 3$

By corollary 6.5.1 neither α_1 nor α_2 can be assigned as a zero-place function. At least one must be a two-place function as $f(a_1,a_2,\ldots,a_n)$ is non-degenerate. The minimum order in this case is 3 . Theorem 6.6 provides the condition for when one of the functions can be assigned to be a one-place function. If such is the case, the decomposition is simple non-disjunctive. If, however, both α_1 and α_2 are true two-place functions the decomposition is complex disjunctive.

A partition containing two sets is more desirable than a partition containing three. Also, when a partition M'' does contain three sets it is most desirable to have $x_{i_q} = x_{j_q}$ for all $x_i , x_j \in N_k \in M''$, $q = 1$ or 2 . The following algorithm is used to find M'' .

Algorithm 6.1

i) If $[M'] = 4$, the decomposition is totally degenerate.

ii) If $X \subset V^2$, each $v_i \in V^2$, $v_i \notin X$ is added to each $S_i \in M'$.

iii) If any pair S_i , $S_j \in M'$ form a cover of X , set $M' = \{S_i , S_j\}$. If there is more than one pair chose the one with the largest number of elements in $S_i \cap S_j$.

iv) If for any i , $S_i = \{x_1,x_2,x_3\}$, $M'' = \{\{x_1,x_2,x_3\}, \{x_4\}\}$.

v) If for any i , $S_i = \{x_1,x_2\}$ and $x_{1_q} = x_{2_q}$, $q = 1$ or 2 , $M'' = \{\{x_1,x_2\} , \{x_3\} , \{x_4\}\}$.

vi) If neither iv) or v) apply, then $M'' = M'$.

3.2 The Assignment of α

Once M'' is found α is assigned by forming a one to one correspondence between the sets of M'' and the elements of W . If M'' contains two sets there are $\frac{4!}{(4-2)!} = 12$ possible mappings, while if $[M''] = 3$, there are $\frac{4!}{(4-3)!} = 24$. We wish to assign α so that the order of the decomposition is minimal. As the decompositions are to be implemented using NAND gates, we also wish to assign α so that the NAND expression for each true two-place α_i has a minimal number of operators.

First consider the case where $[M''] = 2$. By theorem 6.5 , α may be assigned so that α_1 is a zero-place function. The elements 11 and 10 $\in W$ are used as the images of the sets of M'' . Clearly, M'' contains a set of three elements and a set with the fourth element, or M'' contains two sets with two elements in each.

In the first case, α_2 is a vertex function; the minimum NAND representation of which corresponds to assigning the single vertex the image 10 . In the second case, $M'' = \{\{00,11\} , \{01,10\}\}$, the other possibilities leading to degenerate α's . Here $\{00,11\}$ is assigned the image 10 and $\{01,10\}$ is assigned the image 11 . α_2 is the

exclusive OR function which has a simpler NAND representation than the other possibility, the equivalence function.

When $[M'']=3$ the assignment of α is somewhat more complex. α cannot be assigned so that either α_1 or α_2 is a zero-place function, but, by theorem 6.6, if for all $x_i, x_j \in S_\ell \in M''$, $x_{i_q} = x_{j_q}$ then α_q may be assigned to be a one-place function. M'' contains one set of two elements and two sets of a single element each. We denote these sets as $S_k = \{x_1, x_2\}$, $S_m = \{x_3\}$, and $S_n = \{x_4\}$. S_k is assigned the image 11.

Suppose $x_{1_q} = x_{2_q}$. By theorem 6.6, $\alpha_q(a_1,a_2)$ may be assigned to be a one-place function. S_n is assigned the image 00, S_m is assigned the image 01 if $q=1$, or the image 10 if $q=2$. If $x_{1_q} \neq x_{2_q}$ for $q=1$ or 2, S_n is assigned the image 10, and S_m is assigned the image 01.

The above method has the property that whenever possible $\alpha_i(a_1,a_2) = a_i$ or $\alpha_i(a_1,a_2) = \bar{a}_i$. Further, whenever $\alpha_i(a_1,a_2)$ is a true two-place function, it is a vertex function with the single vertex having the image 0. The minimality of a decomposition when $x_{1_q} = x_{2_q}$ depends on which of the two sets of a single element is taken to be S_n. Minimality is achieved if $x_a \in S_n$ has fewer zero co-ordinates than $x_b \in S_m$.

The labeling of the variables of a Boolean function is completely arbitrary. The function

$$f(a_1,a_2,\ldots,a_i,\ldots,a_j,\ldots,a_n)$$

can thus be written

$$f(a_i,a_j,a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_{j-1},a_{j+1},\ldots,a_n) \quad .$$

The above techniques are therefore used to find a minimal NAND expression of a decomposition of $f(a_1,a_2,\ldots,a_n)$ with $A_\lambda = a_i,a_j$, and $A_\mu = a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_{j-1},a_{j+1},\ldots,a_n$.

4. Total Two-Place Decomposition

Ideally, we wish to find the total decomposition of $f(a_1,a_2,\ldots,a_n)$ which corresponds to a minimum cost NAND circuit. Using the conventional multilevel circuit criteria we define:

Definition - The cost of a NAND gate is equal to the number of inputs.

Definition - The cost of a NAND circuit is the sum of the costs of the individual gates.

An exhaustive search of all the possible total decompositions could be performed [2] . Previous authors [14] have used a lexicographic search incorporating the smallest cost previously found as a criterion for eliminating more expensive circuits. The work is reduced as not every total decomposition is examined in full. Even with this refinement, the time required for a complete search is formidable. Partial searchs are often used to save computing time, and consequently it is extremely important that the initial decomposition be very nearly minimal.

4.1 The Selection of Decompositions

There are three types of two-place decompositions:

i) simple disjunctive

ii) simple non-disjunctive

iii) complex disjunctive .

A simple disjunctive decomposition of an n-place Boolean function results in an n - 1 -place image, while the other two types result in an n-place image. A complex disjunctive decomposition, on the other hand, requires more gating to implement than a simple non-disjunctive decomposition. Choosing decompositions in the order described above will thus tend to decrease the number of inputs at successive stages and limit the required hardware.

At a given stage, several decompositions of each type may exist. Two independent criteria have been tried for choosing the best decomposition within the selected type.

i) The decomposition of lowest cost is chosen.

ii) The decomposition whose inputs have come through the fewest number of levels of gating is chosen.

The first one obviously stems from a desire for a minimum total gating cost. The second, on the other hand, will tend to use input variables and gate outputs as soon as possible. This tends to reduce the number of inputs at each successive stage quite rapidly, while also reducing the greatest number of gates between the inputs and output. Rather surprisingly, criterion ii) has been found, through testing, to yield a circuit of lower cost than the first criterion. Hence the second criterion is used.

Once a decomposition is picked the image must be found, and another decomposition chosen. This process continues iteratively until the image is itself a two-place function at which time a total decomposition has been achieved.

4.2 Practical Modifications

The algorithm was initially implemented as described. Several variations were also tested and the following modifications adopted.

It was found that a lower overall cost is achieved if a simple non-disjunctive decomposition is chosen so that the true two-place function has the greater of the two possible costs. This corresponds to assigning α_2 the simplest sum of products expression. Seemingly a contradiction, a theoretical basis for this ad hoc choice is currently under development.

The case where α_i is the exclusive OR function is of simple disjunctive type. The gating required to implement it is however quite costly. A minimal NAND circuit is given below:

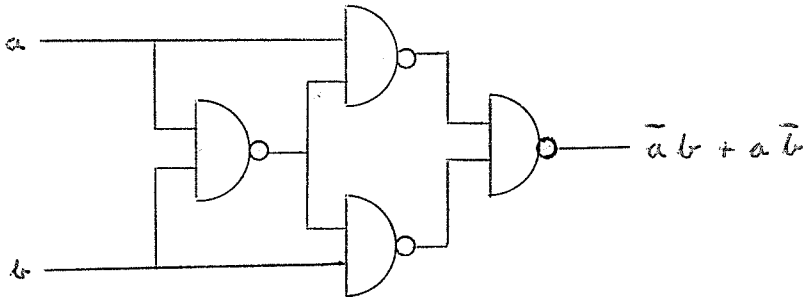


FIG 6.1 NAND implementation of exclusive OR

The cost of this circuit is eight. The level of gating is three. Because of these high values, the method was altered so that the exclusive OR is only used when it cannot be avoided.

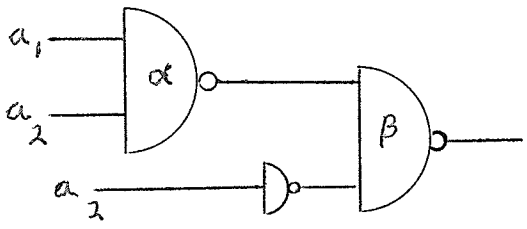
Cycling is a phenomenon in circuits where the output of a gate is:

either i) a constant

or ii) an input variable or its complement

or iii) a previous gate output or its complement .

The algorithm which is used to assign α ensures that when expressed in terms of its inputs each α_i which is actually used is a true two-place function. Consider the circuit



Here $\beta = \overline{\alpha \bar{a}_2}$. However, if the actual expression for α is inserted, then $\beta = a_1 a_2 + a_2 = a_2$. This is a cyclic output.

It was found by inspection that any decomposition involving a previous decomposition and one of its own inputs leads to redundant gating. The exclusion of this type of decomposition has apparently eliminated the cycling problem, as no further problems have been encountered.

In the initial implementation, a new list of decompositions was constructed after each image was found. It was noted that certain decompositions from the previous list reappeared. The possibility of choosing several decompositions at each stage became evident.

Let δ_1 and δ_2 be decompositions given by

$$\delta_1 : f(A) = g(\alpha_1(A_{\lambda_1}), \alpha_2(A_{\lambda_1}), A_{\mu_1})$$

$$\delta_2 : f(A) = h(\beta_1(A_{\lambda_2}), \beta_2(A_{\lambda_2}), A_{\mu_2})$$

Definition - δ_1 and δ_2 are independent if $A_{\lambda_1} \cap A_{\lambda_2} = \phi$, or when $A_{\lambda_1} \cap A_{\lambda_2} \neq \phi$, then either $\alpha_1(A_{\lambda_1})$ and $\beta_1(A_{\lambda_2})$ or $\alpha_2(A_{\lambda_1})$ and

$\beta_2(A_{\lambda_2})$ may be assigned to be one-place functions.

The significance of independence is that the variables required for δ_2 are present in the image of δ_1. Independence is reflexive and intransitive.

Several problems have been examined, and the recurring decompositions have been noted. This has led to the following conjecture.

Conjecture 6.1

If $\{\delta_1,\delta_2,\ldots,\delta_m\}$ is a set of mutually independent decompositions of $f(A)$ a non-degenerate function with

$$f(A) = g(\alpha_1(A_{\lambda_1}),\alpha_2(A_{\lambda_1}),A_{\mu_1})$$

$$f(A) = h(\beta_1(A_{\lambda_2}),\beta_2(A_{\lambda_2}),A_{\mu_2})$$

$$\vdots$$

$$f(A) = k(\gamma_1(A_{\lambda_m}),\gamma_2(A_{\lambda_m}),A_{\mu_m})$$

then there exists a multiple decomposition where

$$f(A) = t(\alpha_1(A_{\lambda_1}),\alpha_2(A_{\lambda_1}),\beta_1(A_{\lambda_2}),\beta_2(A_{\lambda_2}),\ldots,\gamma_1(A_{\lambda_m}),\gamma_2(A_{\lambda_m}),A_{\mu})$$

and $A_{\mu} = A_{\mu_1} \cap A_{\mu_2} \cap \ldots \cap A_{\mu_m}$.

Initially, the list of non-totally degenerate decompositions, whose λ variables are not a previous α_i and one of its inputs, is constructed. The best decomposition is chosen and its image found. Any

decompositions which are not independent of the chosen decomposition are removed from the list, and the next best decomposition is chosen. Decompositions are applied to the image of the previous decomposition. In this way, the image of a multiple decomposition is found. To date no counter-examples to the conjecture have been found.

4.3 The Image of a Decomposition

Algorithms for finding the image of a decomposition have been presented by Griesmer and Karp [33], and Dietmeyer [6]. Although theoretically quite useful, both methods require a great deal of computation.

We are only concerned with two-place decompositions, and only five true two-place functions are allowed. There are only nine possible sub-cubes for the variables of A_λ, and the determination of the image of each cube can be performed by a single table lookup. This table will be given in the description of the computer program.

For decomposition where α_i is a zero-place function, α_i is simply ignored. This follows immediately from the theorem

$$A \cdot 1 = 1 \cdot A = A$$

and the fact that any function may be expressed in normal form. If the zero-place function is the constant 0, the arbitrary assignment of α would have allowed it to be 1, and it can thus be considered redundant.

If α_i is a one-place function, then either $\alpha_i(A_\lambda) = a_i$ or $\alpha_i(A_\lambda) = \overline{a}_i$, $a_i \in A$. Again because of the arbitrary assignment of α we need only consider $\alpha_i(A_\lambda) = a_i$. This is the case where a_i appears in the image of the decomposition. Obviously, a_i is not removed and

it is not necessary to construct α_i . Only true two-place functions need be constructed, and as this is done by table lookup, very little computation is required.

A special case occurs when $\alpha_i(A_\lambda) = a_i$ and the i^{th} - co-ordinate of a given cube is x . The image of the true two-place function is also x . However, these x's in the image cube imply that both co-ordinates of A_λ in the original cube were x , which may not be the case. This problem is avoided by splitting the cube into two cubes, one with the i^{th} co-ordinate 1 , the other with the i^{th} co-ordinate 0 . The problem does not arise if $a_i \varepsilon A_\lambda$ is redundant under the decomposition.

5. The Computer Program

An implementation of the methods presented above has been written in FORTRAN IV for use on an IBM 360/65 computer. Emphasis has been placed on developing efficient computing techniques, but little or no optimization has been attempted. The methods used are machine independent, although the machine used has obviously affected their programming. The techniques are also independent of the FORTRAN IV language, and could be implemented in any suitable language such as ASSEMBLER, PL/1, BASIC or ALGOL. One strict requirement for an efficient implementation is that a logical AND of two bit strings is available in the chosen programming languages.

5.1 The Representation of Cubic Complexes

A method is required for the storage of covers of cubical complexes. In addition to the storage requirements, consideration must be given to the ease of implementing our methods within the chosen representation.

Each cube is represented by a bit-string with two bits for each co-ordinate. The coding used is:

$$00 \Longleftrightarrow \phi$$
$$01 \Longleftrightarrow 0$$
$$10 \Longleftrightarrow 1$$
$$11 \Longleftrightarrow x$$

A minimum of space is required for each cube. It is advantageous for each cube to be individually accessible, and thus a full word (32 bits) is used for each. The bit string representing the cube is right justified, and all unused bits are set to zeros.

Besides compactness, the principle advantage is the simplicity with which the intersection of cubes can be performed. The intersection of the individual co-ordinates of two cubes is represented by the logical AND of their full word representations. If this result contains 00 within the bits allotted to a single co-ordinate, the entire result is ϕ. If it does not, it is the full word representation of the intersection of the cubes. Examples will illustrate.

Example 6.1

CUBES	BIT STRINGS
$11x \cap 110 = 110$	$101011 \cdot 101001 = 101001$

The bit string result is the representation of the required cube 110.

Example 6.2

CUBES	BIT STRINGS
$11x \cap 101 = 1\phi1 = \phi$	$101011 \cdot 100110 = 100010$

Bits 3 and 4 allotted to the second co-ordinate are both zero, and the result as required is ϕ .

The notation of a cube in a full word also facilitates the determination of single co-ordinates, or subcubes, and the alteration of the cube or addition of co-ordinates. The logical AND, or OR, and integer division, and multiplication are used to manipulate the bit strings in the required manner. The use of arithmetic operations requires that the sign bit not be used in the representation of cubes. Each cube is therefore restricted to a maximum of fifteen co-ordinates.

5.2 Programming Techniques

Although the theory was developed using separate ON and OFF covers, the program uses a single function array. Each row consists of a cube and a co-ordinate specifying the corresponding functional value. This system was chosen as it will ease the transition to Dietmeyer's notation of the multiple-output problem [6] , and is more easily extended to multi-valued logic [26] than the two cover scheme. The methods are equivalent as the functional co-ordinate indicates within which cover a cube is contained.

Input to the program is thus a single array of 0's , 1's , and x's which specifies the function to be decomposed. The program is iterative, each step being a multiple decomposition of the current function. As each decomposition is performed, the NAND circuitry

required to implement it is described. The image of each multiple decomposition becomes the specified function to the next step, and the process is repeated until the image is a one-place function. A block diagram of the program is given in Fig. 6.2 .

5.3 Determining the List of Decompositions

Each pair of variables is, in turn, taken to be A_λ . If the variables are a previous α_i and one of its inputs, the pair is ignored. Otherwise, the following algorithm is used to find the simplest decomposition.

Let C be the specification array of the function, and let $c_i \in C$ be the i^{th} row. Denote the co-ordinates of c_i corresponding to A_λ as c_{i_λ} , those corresponding the A_μ as c_{i_μ} , and the functional value co-ordinate as c_{i_f} . A 4×4 compatibility table T is initially filled with ones as we assume each pair of vertices is compatible until an incompatibility is found.

Algorithm 6.2

For each c_i , $c_j \in C$

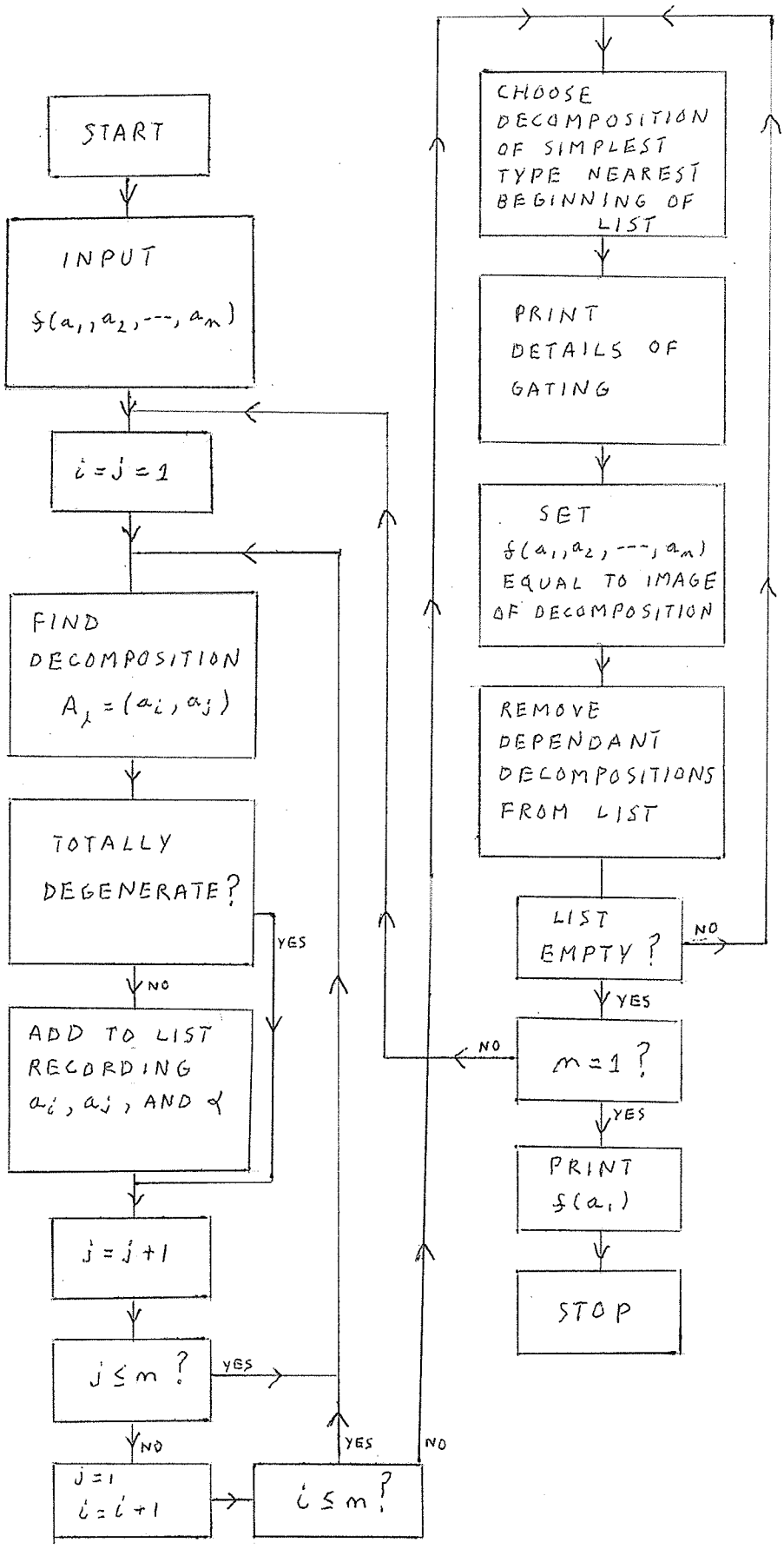
i) If $c_{i_f} = c_{j_f}$, ignore the pair.

ii) If $c_{i_\mu} \cap c_{j_\mu} = \phi$, ignore the pair.

iii) For each vertex $v_k \subseteq c_{i_\lambda}$ and $v_\ell \subseteq c_{j_\lambda}$ set $T_{k_\ell} = T_{\ell_k} = 0$.

There are 64 possible two-place compatibility tables uniquely identified by the six-tuple $\{T_{1_2}, T_{1_3}, T_{1_4}, T_{2_3}, T_{2_4}, T_{3_4}\}$. The decompositions implied by each table have been determined as described above,

FIGURE 6.2


START
INPUT
$f(a_1, a_2, \ldots, a_m)$
$i = j = 1$
FIND DECOMPOSITION
$A_j = (a_i, a_j)$
TOTALLY DEGENERATE?
YES
NO
ADD TO LIST RECORDING
a_i, a_j, AND α
$j = j + 1$
$j \leq m$?
YES
NO
$j = 1$
$i = i + 1$
$i \leq m$?
YES
NO
CHOOSE DECOMPOSITION OF SIMPLEST TYPE NEAREST BEGINNING OF LIST
PRINT DETAILS OF GATING
SET $f(a_1, a_2, \ldots, a_m)$ EQUAL TO IMAGE OF DECOMPOSITION
REMOVE DEPENDANT DECOMPOSITIONS FROM LIST
LIST EMPTY ?
NO
YES
$m = 1$?
NO
YES
PRINT $f(a_1)$
STOP

and the program uses the above six-tuple as an index into the list of these results. In this manner, a great deal of redundant computation is avoided.

5.4 Choosing the Best Decompositions

The variables $\{a_1,a_2,\ldots,a_n\}$ are uniquely associated with the input co-ordinates of the function array specifying $f(a_1,a_2,\ldots,a_n)$. A_λ is in turn taken to be $\{a_1,a_2\},\{a_1,a_3\},\ldots,\{a_1,a_n\},\{a_2,a_3\},\ldots,\{a_{n-1},a_n\}$. For all A_λ which are not a previous α_i and one of its inputs, a decomposition is found and, if it is non-totally degenerate, it is added to the end of the list of decompositions. The decompositions are chosen as described above using the first allowable decomposition encountered. This ordering tends to fulfill selection criterion two. New variables are added to the right end of the function array and by choosing the first decomposition allowed, we tend to use the variables created earliest in the sequence of decompositions.

Associated with each variable is a check bit. When the list of decompositions is constructed, each of these bits is set to 0. As decompositions are selected, the check bits of redundant variables are set to 1. A decomposition is applicable to the image of a previous decomposition only if both check bits for the variables of A_λ are 0. This is logically equivalent to removing dependent decompositions as each decomposition is chosen,but is much simpler to implement.

5.5 Determining the Image of a Decomposition

It was shown that only the true two-place $\alpha_i\ (A_\lambda)$ need be determined explicitly. Any zero-place α_i are ignored, and one-place

α_i are simply input variables which need not be reconstructed. The table used in finding these images is given below.

FUNCTION \ INPUT SUBCUBE	00	01	0x	10	11	1x	x0	x1	xx
$\bar{A}\bar{B}$	0	1	x	1	1	1	x	1	x
$\bar{A}B$	1	0	x	1	1	1	1	x	x
$A\bar{B}$	1	1	1	0	1	x	x	1	x
AB	1	1	1	1	0	x	1	x	x
$A\bar{B} + \bar{A}B$	0	1	x	1	0	x	x	x	x

Each row of the function array is treated separately as previously described. Additional cubes, resulting from the splitting of cubes with an x in the co-ordinate specifying an irredundant variable, are added to the end of the array and their images are found as they are encountered.

Once the image array has been found, subsumed rows are removed. An algorithm using optional products to reduce the number of rows is currently under development. At present, it appears to yield solutions which are usually irredundant and quite often minimal. The use of such an algorithm reduces the computation involved in finding the compatibility tables at the next stage.

Chapter 7

Results and Discussion

1. Introduction

The sample problems presented below were chosen because they represent the general form of the circuits produced by our algorithm. In particular, the examples of Chapters 4 and 5 were included. This will allow an evaluation of our algorithm in terms of previous techniques.

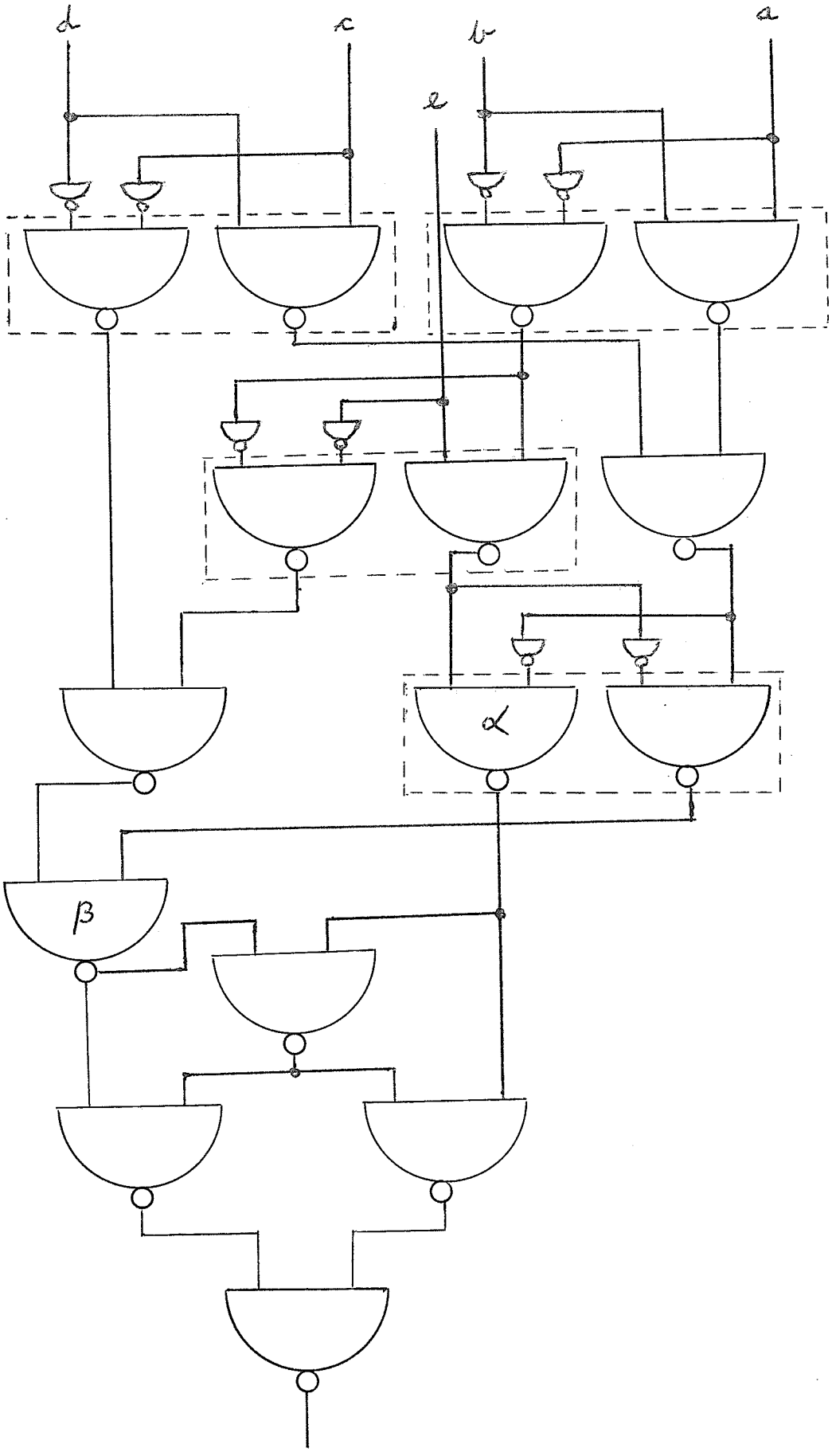
In order to compare algorithms, criteria must be established which determine the relative merits of the circuits produced. The principle criteria is cost. As before, the cost of a circuit is taken to be the total number of gate inputs. In addition, fan-in, fan-out, superfluous gating, and response time must also be considered.

Each of the examples below will be evaluated using these criteria and compared to the circuits produced by previous techniques. Several interesting improvements and extensions to the algorithm will also be considered.

2. Results

Our first example is the two out of five checker of figure 4.5 . Figure 7.1 shows the circuit realized by our algorithm. This result, up to, and including, the gates designated α and β , is equivalent to the result found by Karp et al [14] , figure 5.1 . The final gating is somewhat simpler due to the fact that we have allowed the exclusive OR function. As before, the result is far less expensive than the result found by factoring, figure 4.6 .

FIGURE 7.1



Of particular interest in this example, is the frequency of complex disjunctive decompositions. The pairs of gates resulting from these have been indicated. A method, such as that due to Barnard and Holman [2] , which is restricted to simple decompositions would not yield any result for this circuit. This type of decomposition has been found to be quite prevalent even though it is only chosen when a simple decomposition is not available. As was the case with the result due to Karp et al , our circuit is less costly than the circuit actually used on the IBM 7090 computer and has a lower fan-in requirement.

The example of figure 7.2 was included as it represents the closest comparison found between the decomposition and factoring algorithms. Here the costs were 23 by factoring, and 20 by decomposition. In all other problems tried, the difference was found to be greater. The decomposition circuit has a lower level of gating, and hence a faster response time. In addition, the gating level is more balanced throughout the circuit ranging from 5 to 6 , whereas the level in the factoring circuit ranges from 4 to 7.

The gate designated α in the decomposition result is an example of a factor being utilized both affirmed and negated. As discussed in Chapter 4, this is the main drawback of the factoring approach. The output of the gate β in the factoring result is

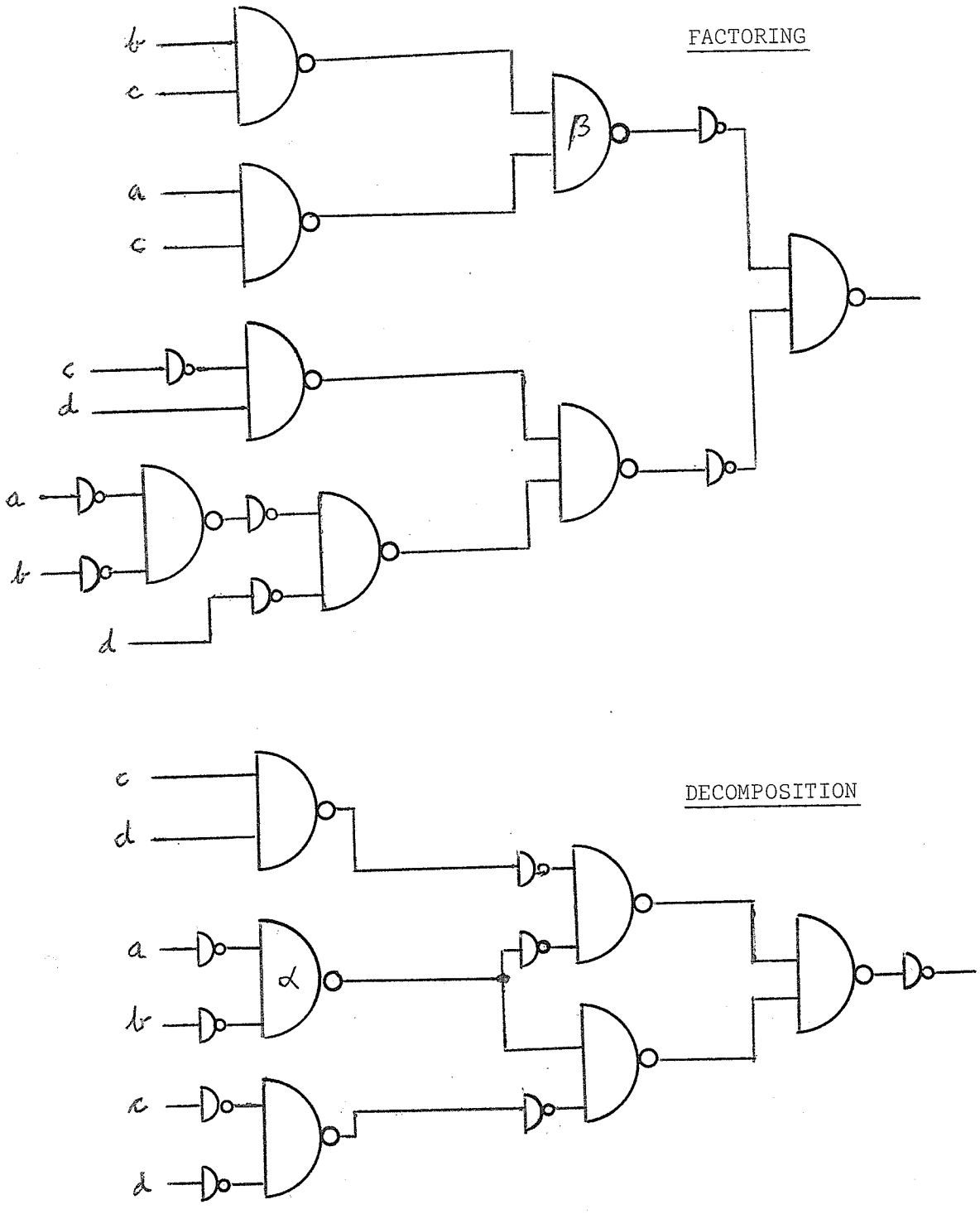
$$ac + bc$$

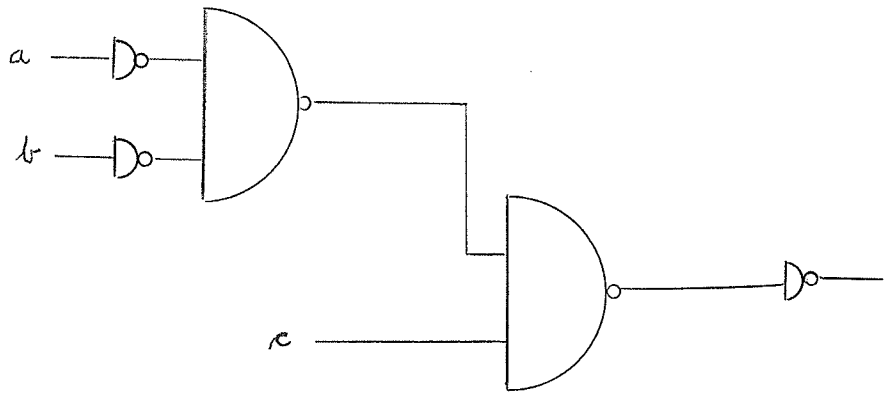
which could be written

$$(a + b)c$$

and realized by the gating

FIGURE 7.2





If this change were made, the resulting circuit would have a cost of 18 . The reason why our decomposition algorithm did not yield this less costly circuit is that a decomposition in the variables c and d was chosen at the first stage. While this was not an optimal choice here, extensive testing has shown that using variables and gates in decompositions as soon as possible, generally leads to less expensive results.

As a final comparison to previous decomposition algorithms we have included the example of figure 5.2. Our result is shown in figure 7.3. Happily, our result is equivalent to that of Karp et al, their three-input NAND gate being replaced by the cascade denoted α and β . This cascade indicates a method for handling multiple-input gates which we shall discuss later. In this example, several of the decompositions were of simple non-disjunctive type. While previous decomposition algorithms have required special techniques for decompositions of this type, our method handles them quite simply. In an optimized computer program, this should result in a substantial reduction in computing time.

The circuit in figure 7.4 is a realization of the function which has a minimal disjunctive normal form of

$$\bar{a}\bar{c}\bar{d} + \bar{b}\bar{c}d + abd + \bar{b}c\bar{e} + \bar{b}\bar{d}e + b\bar{c}\bar{e} + abc + c\bar{d}\bar{e} \quad .$$

FIGURE 7.3

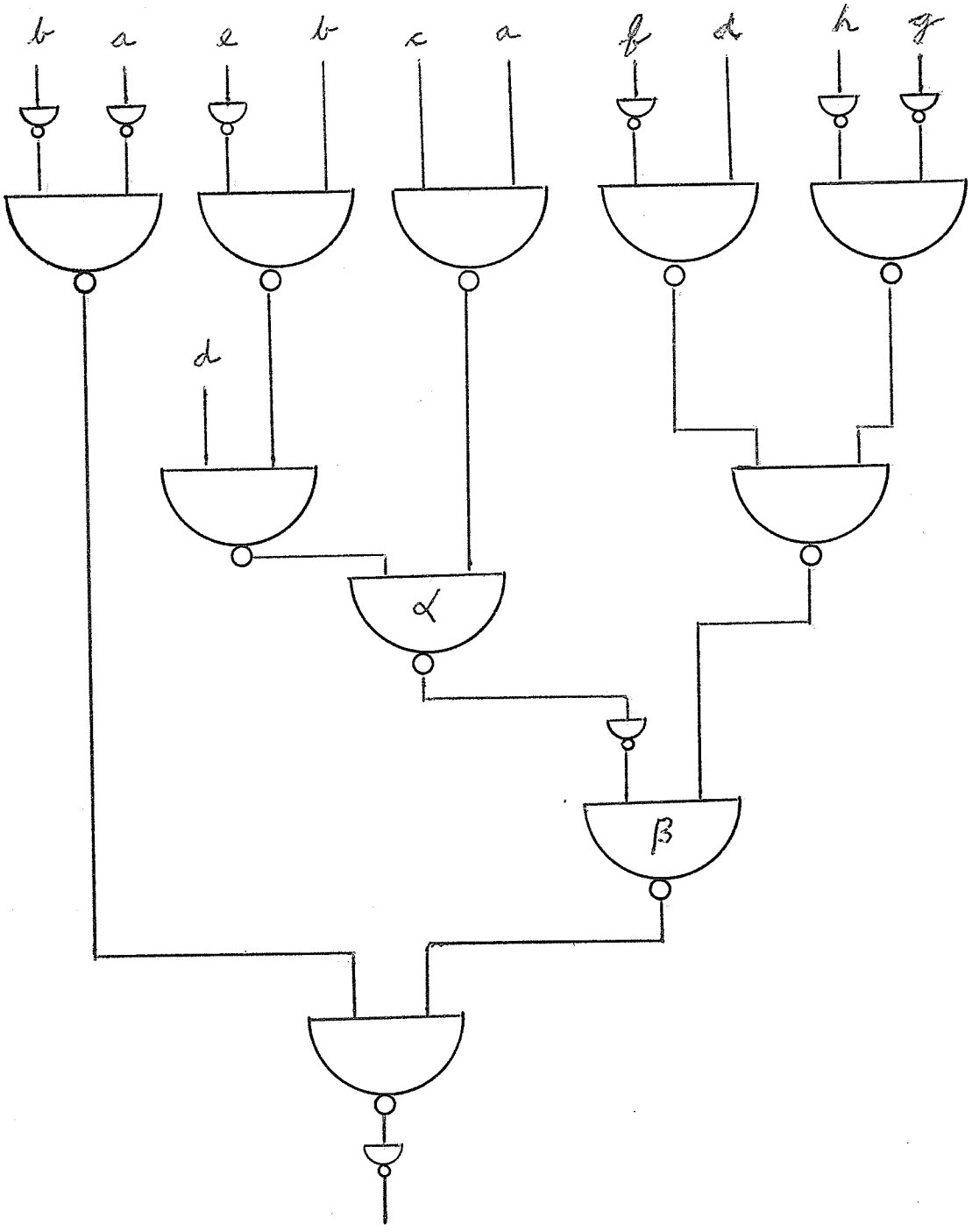
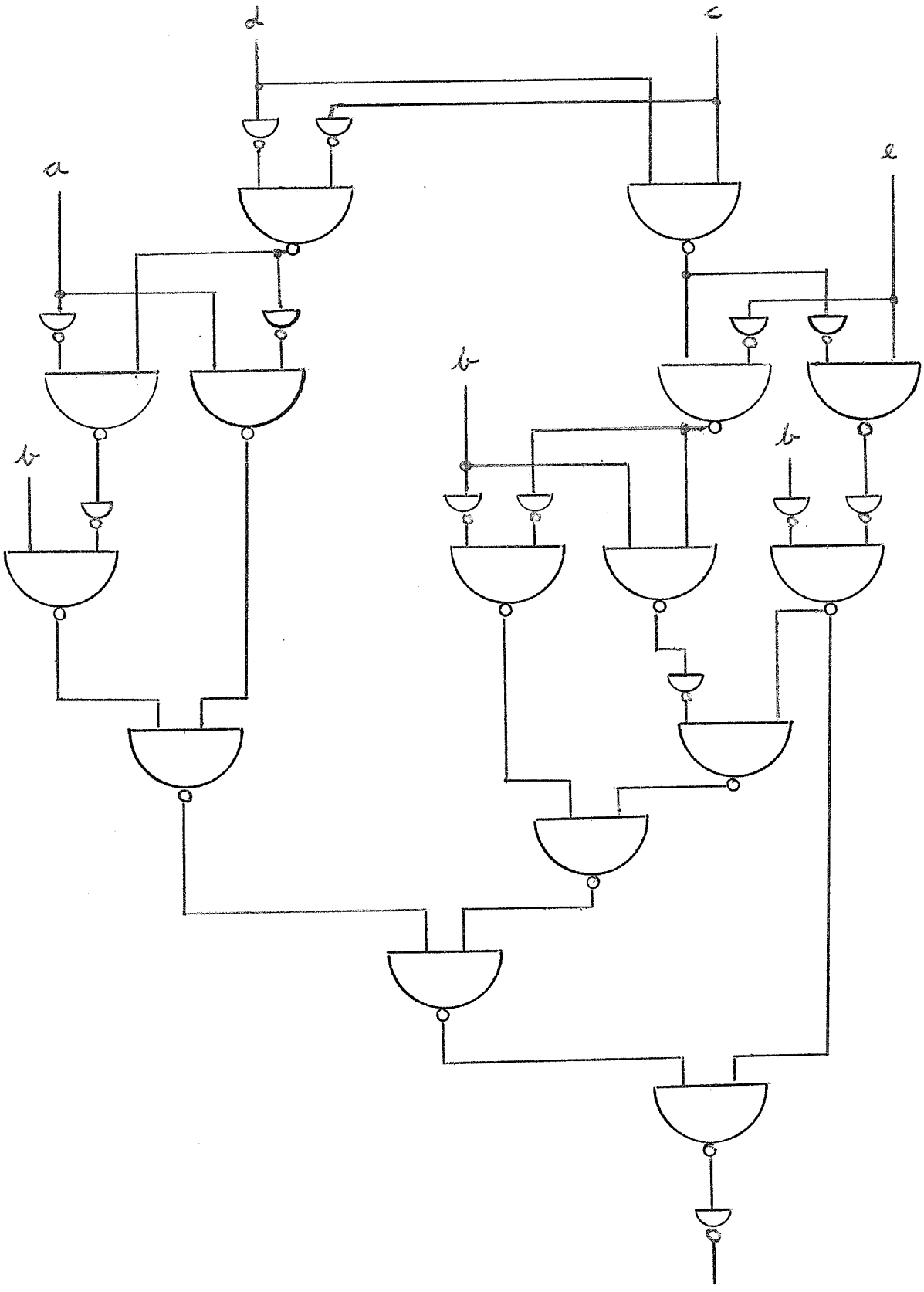


FIGURE 7.4



A three-level NAND realization would have a cost of 37 . Our result has a cost of 43. The overriding factor here is the fan-in limit. In our result this is of course two, but in a three-level realization eight three-input and a single eight-input gate are required. Two-input gates are generally much more commonly used.

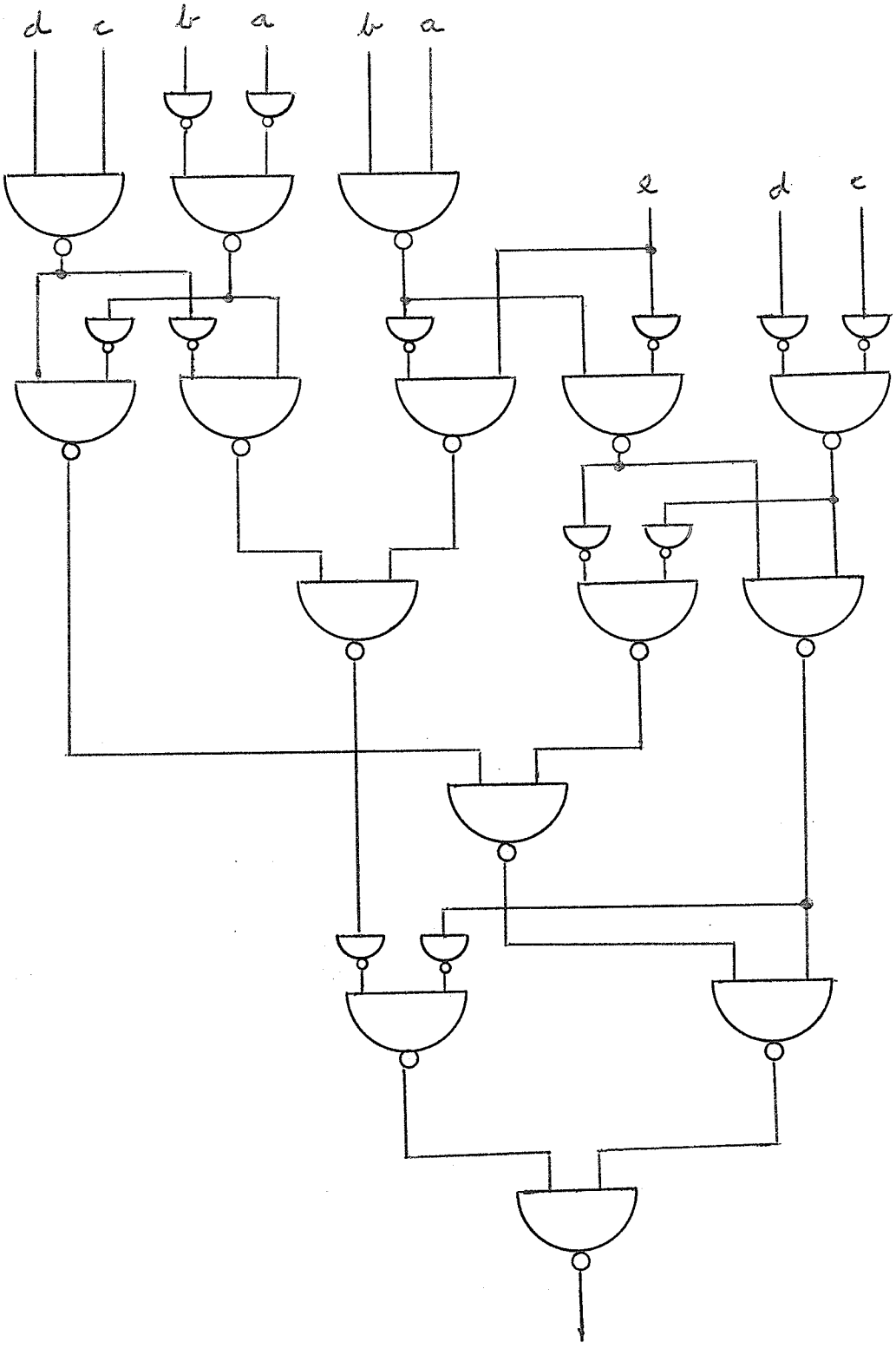
Two points are of interest. First, the variable b appears as a distinct input three times. To date, this has been the highest fan-out required. This value is well within the limits set by gate manufacturers. It thus appears, that while we did not treat it as a definite problem, our algorithm does not generate fan-out difficulties. The second point of interest is the computation involved. In using Quine's first method to find the minimal normal disjunctive form, the prime implicant table became cyclic several times, and the computation was quite involved. A reasonably efficient computer program required about 6.5 seconds on an IBM 360/65 . Our test program, on the other hand, required 3.3 seconds to generate the circuit in figure 7.4 from a minterm specification.

As a further test of the practicality of the algorithm, the function with a minimal disjunctive normal form of

$$ab\bar{d}\bar{e} + a\bar{b}c\bar{e} + a\bar{c}d\bar{e} + a\bar{c}\bar{d}e + \bar{b}cd\bar{e} + \bar{b}\bar{c}de + \bar{a}\bar{b}cd + \bar{a}bc\bar{e} + \bar{a}b\bar{d}e + \bar{a}\bar{b}cd$$

was tried. This is another problem which is extremely involved when solved using Quine's method. Our result is shown in figure 7.5 . Here the cost is 42 . A minimal three-level realization would cost 55 , and would require ten four-input and a single ten-input gate. The improvement is obvious. The computation times were comparable to those above.

FIGURE 7.5



3. Discussion

From these examples, and many others we have tried, it is clear that our algorithm results in as good, or better, circuits than those found by previous techniques. The computer program, while not itself optimal, clearly indicates that the algorithm is easily implemented. The computation required is quite acceptable, and should improve even further with an optimized program. Indications are, however, that still more extensive testing is required before the algorithm can be finalized.

Consider the example in figure 7.6 . Clearly, the program result is not acceptable. What appears in this simple example to be minor extravagance would lead to a gross waste if accumulated over a large circuit. The problem arises from an oversight in the criteria used in the selection of decompositions.

The following criteria are used in the selection of decompositions:

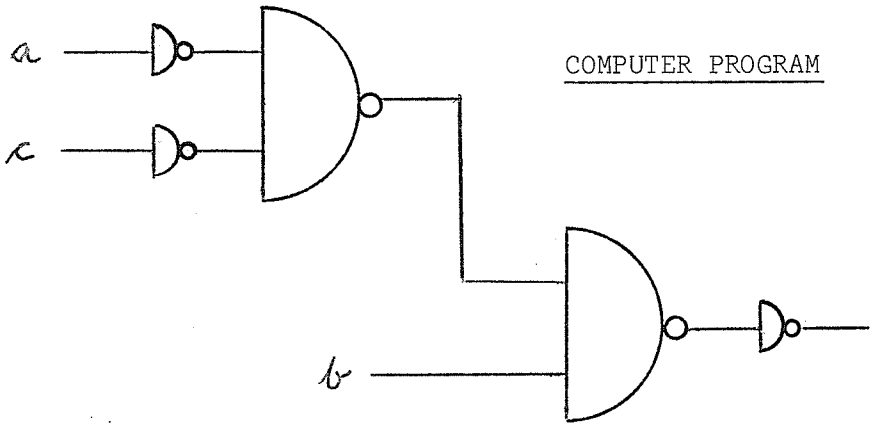
i) choose a simple disjunctive decomposition before either of the two other types;

ii) choose a simple non-disjunctive decomposition before a complex disjunctive decomposition;

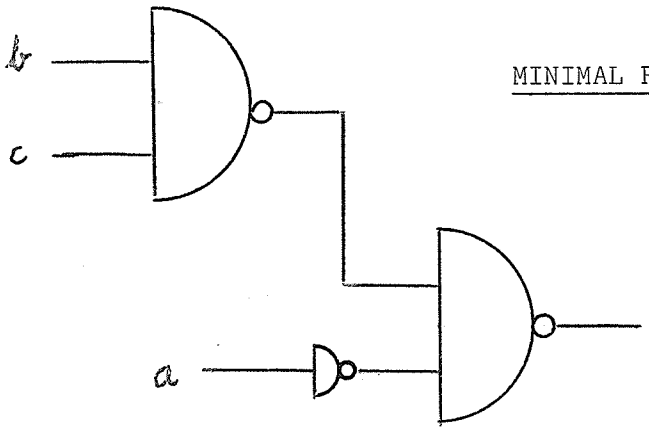
iii) within these limits, choose the decomposition whose inputs have come through the lowest level of gating.

The actual method of implementing these criteria was discussed in Chapter 6 . The oversight involves the case where decompositions of the same type involve variables which have come through the same gating level. Clearly, the decomposition which requires the minimum gating should be chosen. This, however, is not the case, and as shown in figure 7.6 superfluous gating is the result. Implementing a revised selection algorithm would

FIGURE 7.6



COMPUTER PROGRAM



MINIMAL RESULT

require that the level of gating through which each variable has come must be recorded. The change in program logic would be considerable, but the improved results should easily justify this alteration.

An extension of particular interest is the possible use of gates with a fan-in greater than two. This may be done in two ways. First, partitions where A_λ contains more than two variables could be considered. Unfortunately, it was shown in [20] that when X, the domain of α, has more than four elements, the compatibility techniques do not always apply.

A more promising approach would be to examine the final circuit for simplifications. In this way, gates α and β of figure 7.3 could be combined to form the three-input gate of figure 5.2. This is the inverse of the factoring technique of Chapter 4. Additional simplifications may also be possible using the NAND algebra and simplification theorems of Muzio [25]. This approach has the advantage that decompositions will not be absorbed into larger decompositions until their own usefulness has been exhausted.

While NAND gates are quite common, gates of other types are also frequently used. It would thus be useful to have the type of gate being used variable. At present, this only affects the assignment of α. If, however, the cost of a decomposition was incorporated into the selection algorithm, it too would be affected by the type of gate being used. Both of these could easily be controlled by program input and the algorithm is thus easily generalized.

To date, no significant results have been presented on the decomposition of multiple-output functions. The problem is to maximize the gating common to the realizations of the outputs. Karp [13] has shown

that considering $f : V^n \to V^m$ is of little use. The most promising approach appears to be to first find the decompositions of each output function independently. A subset is then chosen with preference given to decompositions arising from more than one output function. An algorithm for this approach is currently under consideration.

Once a reasonable multiple-output algorithm is developed, the decomposition techniques should prove very useful in the design of sequential circuits. The excitation table of a sequential switching circuit provides the necessary specification of the multiple-output function. Other possible applications are the design of cellular logic [21] , and the design of multi-valued switching circuits [26] .

4. Conclusion

The principle disadvantage of the decomposition algorithm is the heuristic approach to choosing decompositions. These methods will only be justified by further extensive testing. An interesting problem would be the development of a theory for constructing total decompositions from the set of two-place decompositions. An approach similar to that taken by Curtis [5] might be tried.

Despite this drawback, the algorithm has been found to yield quite reasonable circuits. The approach to take might possibly be to use this algorithm to obtain a good first approximation, and to then apply simplification rules to improve the circuit. The falling cost of the hardware is steadily doing away with the need for minimal results. What is now needed in the field of logic design are rapid techniques which obtain inexpensive practical circuits. Our algorithm is a good first step in this direction.

References

[1] R.L. Ashenhurst, *"The Decomposition of Switching Functions"*, Annals. Harvard Comp. Lab., Vol. 29, pp. 74-116, 1959.

[2] D.F. Barnard, and D.F. Holman, *"The Use of Roth's Decomposition Algorithm in Multi-Level Design of Circuits"*, Computer Journal, Vol. 11, pp. 269-276, 1968.

[3] T.C. Bartee, *"Computer Design of Multiple Output Logic Networks"*, IRE Trans. Elec. Comp., Vol. EC-10, pp. 21-30, 1961.

[4] A.K. Choudhury, and S.R. Das, *"Determination of One of the Minimal Solutions of Switching Functions"*, Int. J. Control, Vol. 1, pp. 556-583, 1965.

[5] H.A. Curtis, *"A New Approach to the Design of Switching Circuits"*, Van Nostrand, Princeton, New Jersey, 1962.

[6] D.L. Dietmeyer, *"Logic Design of Digital Systems"*, Allyn and Bacon, Boston, Mass., 1971.

[7] R. Fridshal, *"The Quine Algorithm"*, Summaries of Talks at the Summer Institute of Symbolic Logic, Cornell University, pp. 211-212, 1957.

[8] M.J. Ghazala, *"Irredundant Disjunctive and Conjunctive Forms of a Boolean Function"*, IBM Res. & Dev., Vol. 1, pp. 171-176, 1957.

[9] Harvard Computational Laboratory, *"Synthesis of Electronic Computing and Control Circuits"*, Harvard University Press, Cambridge, Mass., 1951, Chapter 5.

[10] S.L. Hight, *"Complex Disjunctive Decomposition of Incompletely Specified Boolean Functions"*, IEEE Trans. Comp., Vol. C-22, pp. 103-110, 1973.

[11] S.T. Hu, *"Mathematical Theory of Switching Circuits and Automata"*, Univ. California Press, Berkeley, 1968.

[12] M. Karnaugh, *"The Map Method for Synthesis of Combinational Logic Circuits"*, Trans. AIEE, Vol. 72, pp. 593-598, 1953.

[13] R.M. Karp, *"Functional Decomposition and Switching Circuit Design"*, J. Soc. Indust. Appl. Math., Vol. 11, pp. 291-335, 1963.

[14] R.M. Karp, F.E. McFarlin, J.P. Roth, and J.R. Wilts, *"A Computer Program for the Synthesis of Combinational Switching Circuits"*, Proc. 2nd AIEE Symp. on Switching Circuit Theory and Logic Design, pp. 152-162, 1961.

[15] E. Kjelkerud, *"A Computer Program for the Synthesis of Switching Circuits by Decomposition"*, IEEE Trans. Comp., Vol. C-21, pp. 568-573, 1972.

[16] R.E. Knispel, *"Boolean Simplification Methods"*, M.Sc. Thesis, University of Manitoba, 1971.

[17] Z. Kohavi, *"Switching and Finite Automata Theory"*, McGraw-Hill, New York, 1970, Chapter 4.

[18] F. Luccio, *"A Method for the Selection of Prime Implicants"*, IEEE Trans. Elec. Comp., Vol. EC-18, 1966.

[19] E.J. McCluskey, *"Minimization of Boolean Functions"*, Bell System Tech. J., Vol. 35, pp. 1417-1444, 1956.

[20] D.M. Miller, and J.C. Muzio, *"Compatibility Techniques for the Decomposition of Ternary Functions"*, Scientific Report No. 71, University of Manitoba, 1973.

[21] D.M. Miller, and J.C. Muzio, *"Decomposition Techniques for Akers Logic Array"*, International Memorandum, University of Manitoba, 1973.

[22] R.E. Miller, *"State Reduction for Sequential Machines"*, IBM Report RC-121, 1959.

[23] G.A. Montgomerie, *"Sketch for an Algebra of Relay and Controller Circuits"*, J. IEE, Vol. 95, pp. 303-312, 1948.

[24] P. Morreale, *"Partitioned List Algorithms for Prime Implicant Determination from Canonical Forms"*, IEEE Trans. Elec. Comp., Vol. EC-16, pp. 611-620, 1967.

[25] J.C. Muzio, *"Gate Reduction Theorems for NAND Circuits"*, Scientific Report No. 62, University of Manitoba, 1972.

[26] J.C. Muzio, and D.M. Miller, *"Decomposition of Ternary Switching Functions"*, Proc. 1973 International Symposium on Multiple-Valued Logic, University of Toronto, pp. 156-165, 1973.

[27] N. Necula, *"A Numerical Procedure for the Determination of the Prime Implicants of a Boolean Function"*, IEEE Trans. Elec. Comp., Vol. EC-16, pp. 687-689, 1967.

[28] S.R. Petrick, *"A Direct Determination of the Irredundant Forms of a Boolean Function from the Set of Prime Implicants"*, Tech. Rep. No. 56-110, AF Cambridge Research Center, Bedford, Mass., 1956.

[29] I.B. Pyne, and E.J. McCluskey, *"The Reduction of Redundancy in Solving Prime Implicant Tables"*, IRE Trans. Elec. Comp., Vol. EC-11, pp. 473-482, 1962.

[30] W.V. Quine, *"The Problem of Simplifying Truth Functions"*, Am. Math. Monthly, Vol. 59, pp. 521-531, 1952.

[31] W.V. Quine, *"A Way to Simplify Truth Functions"*, Am. Math. Monthly, Vol. 62, pp. 627-631, 1955.

[32] J.P. Roth, *"Algebraic Topological Methods for the Synthesis of Switching Systems I"*, Trans. Amer. Math. Soc., Vol. 88, pp. 301-326, 1958.

[33] J.P. Roth, *"Minimization over Boolean Trees"*, IBM Res. & Dev., Vol. 4, pp. 543-558, 1960.

[34] J.P. Roth, and R.M. Karp, *"Minimization over Boolean Graphs"*, IBM Res. & Dev., Vol. 6, pp. 227-238, 1962.

[35] J.P. Roth, and R.G. Wagner, *"Algebraic Topological Methods for the Synthesis of Switching Systems, Part III, Minimization over Nonsingular Boolean Trees"*, IBM Res. & Dev., Vol. 4, pp. 326-344, 1959.

[36] C.E. Shannon, *"A Symbolic Analysis of Relay and Switching Circuits"*, Trans. AIEE, Vol. 57, pp. 713-723, 1938.

[37] T. Singer, *"Some Uses of Truth Tables"*, Proc. Int. Symp. Theory of Switching, Harvard University Press, Cambridge, Mass., pp. 125-133, 1959.

[38] S.Y.H. Su, and D.L. Dietmeyer, *"Computer Reduction of Two-Level Multiple-Output Switching Circuits"*, IEEE Trans. Comp., Vol. C-19, pp. 58-63, 1969.

[39] S.Y.H. Su, and C.W. Nam, *"Computer Aided Synthesis of Multiple-Output Multilevel NAND Networks with Fan-In and Fan-Out Constraints"*, IEEE Trans. Comp., Vol. C-20, pp. 1445-1455, 1971.

[40] E.W. Vietch, *"A Chart Method for Simplifying Truth Functions"*, Proc. Assoc. Comp. Machinery, Richard Rimbach Associates, Pittsburgh, pp. 127-133, 1952.

[41] D. Zissos, and F.G. Duncan, *"Boolean Minimization"*, Comp. J., Vol. 16, pp. 174-179, 1973.