THE UNIVERSITY OF MANITOBA

DECOMPOSITION IN MANY-VALUED LOGIC DESIGN

by

D.M. MILLER

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

WINNIPEG, MANITOBA

MARCH, 1976

"DECOMPOSITION IN MANY-VALUED LOGIC DESIGN"

by

D.M. Miller

A dissertation submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

© 1976

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this dissertation, to the NATIONAL LIBRARY OF CANADA to microfilm this dissertation and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this dissertation.

The author reserves other publication rights, and neither the dissertation nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

TO MAUREEN

ABSTRACT

Decomposition is the reexpression of a function as a composition of simpler functions. The application of decomposition techniques to the logical design of many-valued switching circuits is investigated with particular emphasis on the development of efficient computer-oriented synthesis algorithms.

The principal results which have appeared in the literature are examined. With this basis a theory of two-place decompositions is developed. A technique for identifying the two-place decompositions of a many-valued function is presented. An extremely efficient identification procedure is developed for two-valued functions. Both these techniques apply to either total or partial functions.

The application of two-place decomposition to the synthesis of two-valued and many-valued combinational switching circuits is considered. Algorithms are presented for both single and multiple-output circuits. These algorithms have been implemented on the computer and several sample circuits produced by these algorithms are presented. These circuits will be compared to solutions produced by alternative techniques.

ACKNOWLEDGEMENTS

I wish to express my gratitude to Dr. J.C. Muzio from whom I have learned a great deal and whose supervision was invaluable in the preparation of this thesis. I also wish to thank Drs. F.J. Burkowski, G. Epstein and D.P. Kerr for their constructive criticisms which have improved this thesis. The assistance of Mrs. B. Norman and Mrs. A. Anderson in preparing the typed manuscript is greatfully acknowledged. Financial assistance was provided by the National Research Council of Canada in the form of a fellowship.

TABLE OF CONTENTS

CHAPTER		PAGE
1	Introduction	1
2	Functional Decomposition	12
3	Decomposition of Many-Valued Functions	46
4	A Fast Method for Determining the Two-Place Decompositions of a Two-Valued Function	102
5	Synthesis of Single-Output Circuits	119
6	Synthesis of Multiple-Output Circuits	161
7	Discussion and Conclusion	208
	References	231

CHAPTER 1

Introduction

1. LOGIC DESIGN

The study of the circuits used in computers is a key area of computer science. An appreciation for these circuits and the ways in which they interact with other hardware components is the basis for understanding the limits and capabilities of computers. Despite the development of elaborate software systems which remove most users several levels from the hardware, this area is still central to the development of computer science. For example, paging and segmentation, two major advances in software engineering, can only be implemented with the assistance of special purpose hardware. In addition, the evolution of mini and microcomputers and the subsequent return to 'hands-on' computing, is increasing the hardware knowledge required by programmers.

Hardware can be studied on various levels from the electronic components to the architecture i.e. the way in which units such as memories, central processors, channels, etc. are combined to form a computing system. Between these extremes is logic design. Computer hardware is built from large numbers of a few types of primitive building blocks. Examining the electronic implementation of a block each time it is used would introduce too much detail and only serve to confuse. Logic design presents

a clearer picture by treating the primitive devices in terms of their functional or logical behaviour.

Switching circuits were first constructed of relay contact switches, hence the name. The advent of digital computers in the late 1940's greatly increased the need for speed and resulted in switching circuits being constructed with vacuum tubes. The early 1960's brought a change to discrete component transistor and diode circuits. The last few years have seen incredible advances in integrated circuits. The art is now to the point where a processor can be placed on one chip and sold for less than $200.

Logic design itself covers a broad spectrum. One can talk of building circuits in terms of very primitive devices such as NAND or NOR gates or of interconnecting two chips one of which is a memory the other of which is a processing unit. We shall use logic design to mean design at the gate level. This is still a widely used approach on its own, and is also the basis for integrated circuit design.

2. MANY-VALUED LOGIC DESIGN

Technological limitations have restricted computer manufacturers to the use of two-valued devices. As a result, computer scientists must constantly contend with strings of 1's and 0's and their many associated problems. There is no doubt a machine with truly decimal hardware would be well received. Unfortunately, such a development

does not seem likely. There have, however, been many advances in many-valued hardware implementations especially for the case of three values. Such systems offer several attractive possibilities.

Integrated circuit technology makes the number of interconnections in a circuit the key factor in determining its cost and feasibility. Many-valued devices offer the potential of greatly reducing the number of connections. One study [79] has shown that ternary parallel multipliers require fewer than two-thirds the interconnections of their binary counter-parts. There is a similar reduction in the number of gates but this is offset by their higher cost. The feasibility of integrated ternary devices has been demonstrated by Mouftah and Jordan [43] and by Etiemble and Israel [15].

Another attractive feature of many-valued hardware is symetric number representation as demonstrated by designs presented for ternary arithmetic units [17], [19], [44], [62]. For example, if ternary digits are represented by +1, 0, -1 sign conversion is trivial. Negative values are found by simply inverting each ternary digit i.e. +1 becomes -1, -1 becomes +1 and 0 remains 0. Additions and subtractions are carried out without regard to sign as in the binary two's complement notation. Round-off schemes are not required since a value is rounded to k most significant digits by truncation of the remaining digits.

Switching circuits fall into two categories: combinational where the values of the outputs depend solely on the present inputs; and sequential where the outputs depend both on the present inputs and the previous outputs. Sequential circuits are implemented as a combinational block together with memory devices or feedback loops. Combinational design is thus a more fundamental process. The major part of this thesis examines many-valued combinational circuit design.

3. TWO-VALUED SYNTHESIS TECHNIQUES

Initially, switching circuits were constructed by the heuristic application of ad hoc design techniques. This approach is tedious, highly prone to error and its success depends on the experience of the designer. In 1938, Shannon [64] represented the behaviour of relay contact networks in terms of a two-valued Boolean algebra. Since that time, there has been a great deal of interest in application of Boolean algebra to the synthesis problem (ref [4], [5], [16], [18], [33], [35], [42], [48], [51], [55], [56], [64], [65] etc.). Particular attention has been given to the development of algorithmic design procedures and to their implementation on the computer.

The key results were presented by Quine [55], [56] who developed techniques for determining a minimal disjunctive normal form Boolean expression. Such an expression

is a disjunction of terms each of which is a conjunction of variables or their inversions e.g. $\bar{a}bc + ab\bar{c} + a\bar{b}c\bar{d}$. The expression is minimal if the number of terms plus the number of variables in these terms is as small as possible. These expressions correspond directly to two-level switching circuits with minimal numbers of gates and connections. Several authors have suggested improvements to Quine's techniques (see reference list in preceding paragraph). The most important are the computer implementation due to McCluskey [35] and Bartee's technique [4] for multiple-output circuits.

Zissos and Duncan [83] have presented an alternative technique which is computationally more efficient than the processes based on Quine's work. It is not clear whether this technique always produces a minimal result but the results are typically very good and are obtained quickly. Geometric techniques for two-level circuit design have also appeared e.g. the map methods of Vietch [78] and Karnaugh [28] and the combinatorial topology approach due to Roth [58]. The latter approach was examined in detail in [36].

Two-level minimization techniques for many-valued functions have been presented by Allen and Givone [1], Smith [68], Ostapko _et al_ [50], Su and Cheung [70], [71] and Nutter and Swartout [49]. These techniques require considerable computation and are only practical for very small problems involving few variables.

A minimal two-level circuit may not be optimum. Often the fan-in limit i.e. the maximum number of inputs to a single gate, is too large for practical implementation. In addition, there is frequently a multi-level circuit with a lower fan-in limit, fewer gates and consequently fewer interconnections. Design techniques for multi-level switching circuits are thus extremely important.

One approach is to begin with a minimal two-level circuit and to transform it to an equivalent multi-level circuit. This procedure is often applied by hand in which case the experience of the designer is critical. There is also a high chance of error. Several authors [11], [12], [13], [73] have considered algorithmic procedures involving factoring operations where large gates are replaced by equivalent structures made up of smaller gates. These techniques were specifically designed for computer execution which relieves the designer from the tedium of hand execution and lessens the chance of error.

Factoring has three principle drawbacks. The computation required to find the initial two-level solution can be quite costly. There is no guarantee the resulting circuit will be optimal. Finally, factoring techniques have only been developed for vertex-type gates i.e. gates which assume the value 0 (or 1) for exactly one input condition. They exclude the use of exclusive-OR gates and cannot be extended to many-valued problems since there is no many-valued equivalent to vertex gates.

3. DECOMPOSITION

The most fundamental model of a switching circuit is as a function mapping the possible input assignments onto the values assumed by the output. Decomposition is a reexpression of such a function as a composition of simpler functions. This corresponds directly to realizing the initial switching circuit as a composition of simpler circuits. There has been considerable interest in applying decomposition techniques to the synthesis of two-valued combinational circuits and to a lesser degree, many-valued combinational circuits. The general approach is to identify a sequence of decompositions which express the circuit in terms of continually simpler subcircuits until it is entirely in terms of primitive switching devices.

The first major study of this approach was presented by Ashenhurst [2]. This was extended by Curtis [9] who presented an algorithm for the synthesis of two-valued switching circuits without any unspecified input conditions. An alternative approach was developed by Roth and Karp [59], Karp et al [30] and Karp [29]. This technique is computationally more viable and can be applied to circuits with incompletely specified input conditions which arise frequently in practice. Many-valued decomposition techniques have been considered by Waliuzzaman and Vranesic [82], Varshevsky [77] and by Muzio and the author [38], [45]. To date there have been no results for multiple-output circuits.

Decomposition techniques have several advantages over the other approaches to design. First, decomposition algorithms yield multi-level circuits without requiring an initial two-level solution. In fact since decomposition is a property of the function and not the notation of the function, we are free to choose the notation based on other constraints such as the efficiency of its representation on the computer. Since decomposition treats gates in terms of their functional properties and not the particular functions they implement, it can be adapted to use varying primitive devices. This is extremely important in the many-valued case where there is no agreement as to which are the most viable primitives. Computationally a major advantage of decomposition techniques is that they generate relatively few temporary results. This is important since this has been a drawback to computer-assisted logic design especially for the algebraic techniques.

In this thesis, we examine decomposition techniques for two-valued and many-valued functions and the application of these techniques to the synthesis of combinational switching circuits. Chapter 2 reviews the principal decomposition results which have appeared in the literature. Chapters 3 and 4 examine the problem of determining decompositions of many-valued and two-valued functions respectively. A design algorithm for single-output many or two-valued circuits is developed in chapter 5. This is a re-

finement and extension of the algorithm presented by the author in [36]. Multiple-output circuits are considered in chapter 6. A general algorithm is presented, but for reasons discussed in chapter 6, it has only been implemented for two-valued functions. Chapters 5 and 6 contain several sample circuits produced by our algorithms. These are compared to circuits produced by alternative techniques.

4. NOTATION AND DEFINITIONS

The following general notation and definitions are used throughout the thesis. More detailed notation and definitions will be introduced within the text as required where the context of the discussion will make them more understandable.

We use capital letters as the names of sets and the corresponding small letters with subscripts to denote their elements. $|A|$ will denote the cardinality of the set A i.e. the number of elements in A. $\cup$ will denote set union. $\cap$ will denote set intersection. ϕ will denote the empty set. Small letters are also used as arguments of functions and as variables. The context will determine the use.

Functions are denoted by the letters f,g,h or $\alpha,\beta,\gamma,\ldots$ $\ldots,\delta$. The range of a function will be denoted by R(f), the domain will be denoted by D(f). The range and domain are of course both sets. Functions will be specified in one of three ways:

i) $f(A)$ - where A is a set of arguments,

ii) $f(a_1,a_2,\ldots,a_n)$ - where $a_1,a_2,\ldots,a_n$ denote the arguments,

iii) $f[x]$ - where x denotes an element in $D(f)$.

In some cases two-valued functions are represented by expressions in Boolean algebra. The standard notation is used [21] with + denoting disjunction, $\cdot$ denoting conjunction and inverted variables represented by a bar e.g. $\bar{a}$. The conjunction symbol is ommitted between adjacent variables.

A many-valued function is such that the function and its arguments each may take on only a finite number of values. The function is said to be r-valued if these values are taken from $\{0,1,2,\ldots,r-1\}$ and the function itself or one of its arguments takes on all r values. A two-valued function is the case where r=2. An n-place function has n arguments. It is said to be a true n-place function if it depends on all of its arguments i.e. no arguments are redundant. A function $f(A)$ is total if it is defined for all possible assignments to the a_i; otherwise it is partial and the assignments to the a_i for which f is undefined are termed 'don't-care' conditions.

We shall use the term switching circuit to mean a combinational switching circuit. A two-valued switching circuit is one composed of devices implementing two-valued functions. The following gate symbols are used:

AND

α	b=0	b=1
a=0	0	0
a=1	0	1

OR

α	b=0	b=1
a=0	0	1
a=1	1	1

NAND

α	b=0	b=1
a=0	1	1
a=1	1	0

NOR

α	b=0	b=1
a=0	1	0
a=1	0	0

EXOR
(exclusive-OR)

α	b=0	b=1
a=0	0	1
a=1	1	0

INVERTER

a	α
0	1
1	0

A many-valued switching circuit is one composed of devices implementing many-valued functions. In drawing these circuits all devices are denoted by the AND symbol with a label identifying a truth table defining the required function.

CHAPTER 2

Functional Decomposition

1. INTRODUCTION

Functional decomposition is the re-expression of a function as a composite of functions. For example, the two-valued function

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

may be written

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

where $\delta(\alpha,\beta) = \alpha\cdot\beta$, $\alpha(a,b) = a + b$, and $\beta(c,d) = c + d$. Decompositions where each subfunction $\alpha,\beta,\gamma, \ldots, \delta$ can be realized by a switching function are of particular interest. Such decompositions represent multi-level circuit realizations of the original function.

A number of authors have considered functional decomposition and the application of decomposition techniques to the synthesis of switching circuits. The first major study was presented by Ashenhurst [2]. Decompositions of total two-valued functions were considered. Each function in these decompositions is two-valued and no functions have common arguments. Ashenhurst's work is the basis of nearly all the decomposition results appearing in the literature.

Curtis [9] has presented the most useful generalization of Ashenhurst's results. Once again total two-

valued functions are considered. In this case, each function in the decomposition is two-valued and functions may have common arguments.

Roth and Karp [59] have presented a very general characterization of decomposition. These authors have applied this result to the decomposition of partial two-valued functions. It will be shown that this result can be used to identify a large class of decompositions of partial many-valued functions. This result is the basis of the decomposition techniques developed in chapters 3 and 4.

Two approaches to circuit synthesis employing decomposition techniques have appeared. The first was developed by Ashenhurst [2] and Curtis [9]. Certain 'simple' decompositions are determined by examining the given function. Several theoretical results are then applied to construct a more 'complex' decomposition from the simple decompositions. This approach is applied iteratively until the initial function is expressed entirely in terms of functions corresponding to switching devices.

The results employed in the construction of complex decompositions do not extend to partial or to many-valued functions. The first synthesis approach can thus not be used in these cases.

The second approach was suggested by Roth and Karp [59]. As above simple decompositions are found

by examining the given function. Each of these simple decompositions consists of a number of functions corresponding to switching devices and at most one function to be decomposed further. A circuit realization is found by constructing a sequence of simple decompositions.

In this chapter the principal decomposition results are examined. This material provides both the motivation and the background for the development in later chapters.

2. SIMPLE DISJUNCTIVE DECOMPOSITION

A simple disjunctive decomposition is an expression of the form

$$f(A) = g(\alpha(A_\lambda), A_\mu) \qquad (1)$$

where $A_\lambda \cup A_\mu = A$, $A_\lambda \cap A_\mu = \phi$. The decomposition is termed simple since it involves only a single α and disjunctive since g and α have no common arguments. If f is partial g may be defined when f is not. Whenever f is defined, the equality must hold.

Ashenhurst [2] has considered the problem of determining the simple disjunctive decompositions of a total two-valued function. α is restricted to being two-valued.

Let f(A) be an arbitrary total two-valued function. The sets A_λ and A_μ are a $\lambda\mu$-partition of A. Ashenhurst's technique requires f(A) be represented by a partition matrix. A partition matrix has $2^{|A|}$ elements each of

which is an entry from the truth table defining f. The columns of the matrix are each uniquely labeled with one of the $2^{|A_\lambda|}$ assignments to the variables of A_λ. The rows of the matrix are each uniquely labeled with one of the $2^{|A_\mu|}$ assignments to the variables of A_μ. These labels identify a unique assignment to the variables of A with each position in the matrix. These assignments in turn define the values of the elements.

Consider the function given by the truth table below.

a_1	a_2	a_3	a_4	f(A)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

For the partition $A_\mu = \{a_1, a_3\}$, $A_\lambda = \{a_2, a_4\}$ the corresponding partition matrix is given below.

	a_2a_4	00	01	10	11
a_1a_3	00	0	0	0	0
	01	0	0	0	0
	10	0	1	1	1
	11	1	1	1	1

The <u>column multiplicity</u> of a partition matrix F (denoted γ) is the number of distinct columns in F. The matrix above has 2 distinct columns. Ashenhurst's criterion is a bound on γ.

<u>Theorem 2.1</u> (Ashenhurst [2]) Given f a total two-valued function, there exists a simple disjunctive decomposition

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

if, and only if, the corresponding partition matrix has $\gamma \leq 2$.

In the above example, there is a decomposition $f(A) = g(\alpha(a_2, a_4), a_1, a_3)$.

The partition matrix is also used to explicitly determine α and g. Ashenhurst has shown if $\gamma \leq 2$ the matrix contains at most four distinct rows. Each of these is one of

i) a row containing all 1's;

ii) a row containing all 0's;

iii) a row containing both 0's and 1's;

iv) a row which is the inverse of the row in iii) i.e. it is 1 when the row in iii) is 0 and 0 when the row in iii) is 1.

Our example contains a zero row, a one row and a nonconstant row.

Each row of the partition matrix defines a function of the variables of A_λ. If $\gamma \le 2$ each of these functions is one of

i) the constant function 1;

ii) the constant function 0;

iii) some function $\beta(A_\lambda)$

iv) $\bar{\beta}(A_\lambda)$

α is chosen as either β or $\bar{\beta}$. Once α is chosen g is fixed. Determining g is quite straightforward.

In our example there is a single nonconstant row which is $\beta(a_2, a_4)$. $\beta(a_2, a_4) = 1$ when a_2 or a_4 is 1. $\beta(a_2, a_4)$ is 0 when $a_2 = a_4 = 0$. Choose $\alpha = \beta$.

$g(\alpha(a_2, a_4) a_1, a_3)$ must be 0 whenever a_1 is 0 since the first two rows of the partition matrix are 0. g must be 1 whenever $a_1 = a_3 = 1$. Finally, $g(\alpha(a_2, a_4), a_1, a_3) = \alpha(a_2, a_4)$ when $a_1 = 1$ and $a_3 = 0$. g is given by the following truth table. g is termed the image of the decomposition. In general, the image is the function which results from decomposing the initial function.

a_1	a_3	α	g
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

For an n-variable function there are 2^n partition matrices. Two of these represent the cases where $|A_\lambda| = 0$ or $|A_\lambda| = n$ which are of no use. A further n are the cases where $|A_\lambda| = 1$. These partitions are only useful in determining if the function has a redundant argument i.e. an argument whose value never affects the value of the function. There are thus $2^n - n - 2$ partitions which can represent simple disjunctive decompositions.

Waliuzzaman and Vranesic [82] have extended Ashenhurst's condition to total many-valued functions. In this case α may assume as many values as f. For a given partition A_λ, A_μ, a total r-valued function f(A) has a simple disjunctive decomposition if, and only if, the corresponding partition matrix has $\gamma \leq r$.

For example, the partition matrix given by

		$a_1 a_3$															
		00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
	0	0	2	3	0	2	2	3	0	3	3	3	0	0	0	0	0
a_2	1	1	2	3	1	2	2	3	1	3	3	3	1	1	1	1	1
	2	2	2	3	2	2	2	3	2	3	3	3	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

has 3 distinct columns. $r = 4$ and thus there is a simple disjunctive decomposition

$$f(A) = g(\alpha(a_1, a_3), a_2)$$

where α is 3-valued.

The rows of this matrix do not obey any convenient pattern as they did in the two-valued case. Determining an α is thus somewhat trickier. We shall illustrate the method by determining an α for the matrix above.

The column labels are grouped into sets so that labels on equal columns are in the same set. These sets are {00, 03, 13, 23, 30, 31, 32, 33}, {01, 10, 11} and {02, 12, 20, 21, 22}. Each of these sets is assigned a unique value between 0 and γ-1. α is defined so that each assignment yields the value assigned to the set in which it appears.

For our example one possible α is given below.

		a_3			
	α	0	1	2	3
	0	0	1	2	0
a_1	1	1	1	2	0
	2	2	2	2	0
	3	0	0	0	0

It is easily shown that if a given matrix has $\gamma \le r$, then there are $\frac{r!}{(r-\gamma)!}$ possible choices for α.

Determining g is also more complex than in the two-valued case. When $\alpha(a_1, a_3) = 0$, $f(A) = a_2$ and $g(0, a_2) = a_2$. When $\alpha(a_1, a_3) = 2$, $f(A) = 3$ and $g(2,a_2) = 3$. For $\alpha(a_1, a_3) = 1$, we find g is given by

a_2	$g(1, a_2)$
0	2
1	2
2	2
3	3

Finally we note that α never assumes the value 3 and these conditions are don't-cares. g is given by

		(a_1, a_3)			
		0	1	2	3
a_2	0	0	2	3	-
	1	1	3	3	-
	2	2	2	3	-
	3	3	3	3	-

3. SIMPLE NONDISJUNCTIVE DECOMPOSITION

Simple disjunctive decomposition is rather restrictive since g and α must have no common arguments. The natural extension is to allow g and α to have common arguments. Decompositions of this type are termed simple nondisjunctive. Curtis [9] has presented a method for determining the simple nondis-

junctive decompositions of a total two-valued function.

A simple nondisjunctive decomposition is an expression of the form

$$f(A) = g(\alpha(A_\lambda), A_\mu) \qquad (2)$$

where $A_\lambda \cup A_\mu = A$ and $A_\lambda \cap A_\mu \neq \phi$. The set $A_\lambda \cap A_\mu$ is termed the bound set. The set $A_\lambda - (A_\lambda \cap A_\mu)$ is termed the free set.

Suppose there is a decomposition as in (2) with f, g and α two-valued functions. Let B denote the bound set. For each of the $2^{|B|}$ assignments to the bound variables there are functions f_i, g_i and α_i such that

$$f_i(A - B) = g_i(\alpha_i(A_\lambda - B), A_\mu - B) \qquad (3)$$

f_i, g_i and α_i are found by substituting the assignment to the variables B into f, g and α respectively.

The key to Curtis's result is that (3) is a simple disjunctive decomposition which can be determined as described above. Curtis has shown that a simple nondisjunctive decomposition exists if, and only if, (3) holds for each assignment to the bound variables. To test for a simple nondisjunctive decomposition, a free and a bound set are chosen, and each subfunction corresponding to an assignment to the bound set is tested for a simple disjunctive decomposition. The α_i

together define α; the g_i together define g. Curtis has further shown that with a judicious choice of the row and column labelings of the partition matrices described above, the matrices to test the f_i are easily extracted submatrices.

Curtis's approach is to expand the simple nondisjunctive case to a number of simple disjunctive decompositions. This expansion does not depend on the fact that f is two-valued. It appears Waliuzzaman and Vranesic's criterion can be extended in the same manner.

4. COMPLEX DECOMPOSITION

A complex decomposition is a re-expression of a function as a composition of more than two functions. Complex decompositions are determined by applying several theoretical results to the set of simple decompositions of a function. Curtis [9] and Karp [29] have presented the main results. Complex decompositions have been characterized for total two-valued functions and to a lesser degree for total many-valued functions.

A multiple disjunctive decomposition is an expression of the form

$$f(A) = g(\alpha_1(A_{\lambda 1}), \alpha_2(A_{\lambda 2}), \ldots, \alpha_t(A_{\lambda t}), A_\mu) \quad (4)$$

where $A_{\lambda 1} \cup A_{\lambda 2} \cup \ldots \cup A_{\lambda t} \cup A_\mu = A$, $A_{\lambda i} \cap A_\mu = \phi$, $1 \le i \le t$, and $A_{\lambda i} \cap A_{\lambda j} = \phi$, $1 \le i \le t$, $1 \le j \le t$, and where each α_i assumes no more values than f.

Theorem 2.2. (Karp [29]) Given a total function f(A) there exists a decomposition of the form (4) if, and only if, there exist functions h_i, $1 \le i \le t$, such that

$$f(A) = h_i(A_{\lambda 1}, A_{\lambda 2}, \ldots, \alpha_i(A_{\lambda i}), \ldots, A_{\lambda t}, A_{\mu})$$

for $1 \le i \le t$.

A second form of complex disjunctive decomposition is the iterative disjunctive decomposition which is given by

$$\left.\begin{aligned} \alpha_1 &= \alpha_1(A_{\lambda 1}) \\ \alpha_j &= \alpha_j(\alpha_{j-1}, A_{\lambda j}),\ 2 \le j \le t \\ f(A) &= g(\alpha_t, A_{\mu}) \end{aligned}\right] \qquad (5)$$

where $A_{\lambda 1} \cup A_{\lambda 2} \cup A_{\lambda 3} \cup \ldots \cup A_{\lambda t} \cup A_{\mu} = A$, $A_{\lambda i} \cap A_{\mu} = \phi$, $1 \le i \le t$, $A_{\lambda i} \cap A_{\lambda j} = \phi$, $1 \le i \le t$, $1 \le j \le t$, and where no α_i assumes more values than f.

A simple decomposition

$$f(A) = g(\alpha(A_{\lambda}), A_{\mu})$$

is termed strict if the range of α has as few elements as possible.

Theorem 2.3. (Karp [29]) Given a total function f(A) with decompositions

$$f(A) = h_i(\alpha_i(A_{\lambda 1}, A_{\lambda 2}, \ldots, A_{\lambda i}), A_{\lambda i+1} \ldots, A_{\lambda t}, A_{\mu})$$

for $1 \le i \le t$, where each decomposition except perhaps the first is strict, then for each $1 \le j \le t$ there is a β_j such that

$$\alpha_j = \beta_j(\alpha_{j-1}(A_{\lambda 1}, A_{\lambda 2}, \ldots, A_{\lambda j-1}), A_{\lambda j}) .$$

A more general class of decompositions include multiple disjunctive and iterative disjunctive decompositions as special cases. These <u>disjunctive treelike decompositions</u> have the form

$$\alpha_j = \alpha_j(A_j, B_j), \ 1 \le j \le t,$$
$$f(A) = \alpha_{t+1}(A_{t+1}, B_{t+1})$$

where $A_1, A_2, \ldots, A_{t+1}$ are disjoint subsets of A which together exhaust A and $B_1, B_2, \ldots, B_{t+1}$ are disjoint subsets of $B = \{\alpha_1, \alpha_2, \ldots, \alpha_t\}$ which together exhaust B and where B_i is a subset of $\{\alpha_1, \alpha_2, \ldots, \alpha_{i-1}\}$, $1 \le i \le t+1$. A disjunctive treelike decomposition has the property that no two α_i have a common argument.

The structure of a disjunctive treelike decomposition is defined by the sets A_i, B_i, $1 \le i \le t+1$. Karp [29] has presented an answer to the following question: given the sets A_i, B_i, $1 \le i \le t+1$, and given positive integers r_i, $1 \le i \le t+1$, under what conditions is it possible to specify α_i, $1 \le i \le t+1$, so that

i) the decomposition represents a given total function;

ii) each α_i has at most r_i elements in its range?

Let ε_j denote the variables in A on which the value of α_j depends. In the tree below, $\varepsilon_1 = \{a_1, a_4\}$, $\varepsilon_2 = \{a_3\}$, $\varepsilon_3 = \{a_2, a_8\}$, $\varepsilon_4 = \{a_5, a_6\}$. Since the value of α_5 depends on $\alpha_1, \alpha_2, \alpha_3$, $\varepsilon_5 = \{a_1, a_2, a_3, a_4, a_8\}$, $\varepsilon_6 = \{a_5, a_6, a_7\}$, $\varepsilon_7 = A$.

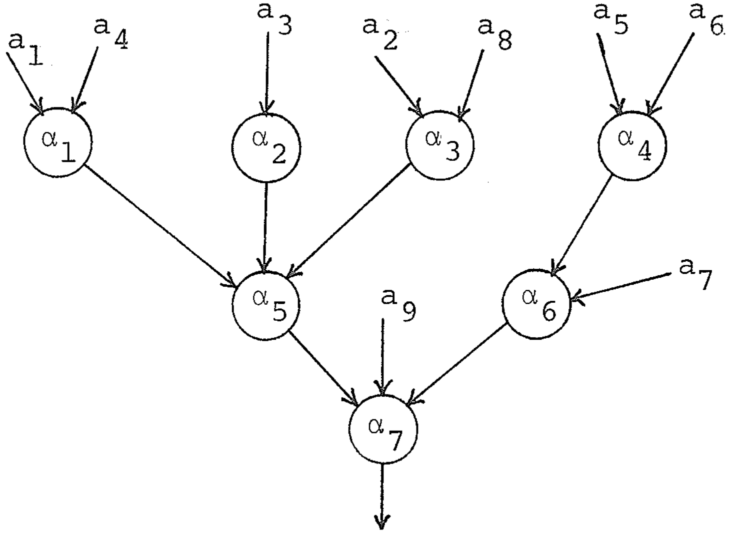


FIGURE 2.1

Let $k(\varepsilon_i, f)$ denote the column multiplicity of the partition matrix representing f whose columns are labeled by the assignments to the variables of ε_i.

Theorem 2.4. (Karp [29]) Given A_i, B_i, $1 \le i \le t+1$, and a total function f, there exist α_i, $1 \le i \le t+1$, with no more than r_i elements in the range of α_i, such that the given decomposition represents f if, and only if, $r_i \ge k(\varepsilon_i, f)$ for all $1 \le i \le t+1$.

The complex decompositions above are constructed from simple disjunctive decompositions. Karp [29] has presented a partial characterization of iterative nondisjunctive decompositions of total many-valued functions i.e. iterative decompositions constructed from simple nondisjunctive decompositions. Except for this isolated result nothing is known on the complex decomposition of many-valued functions.

Curtis [9] has considered the proper complex decomposition of total two-valued functions. A proper complex decomposition is one composed of simple decompositions both disjunctive and nondisjunctive. Even a cursory review of these results would be quite lengthy. Since they are strictly two-valued results and thus not directly relevant to this thesis they are omitted.

5. DECOMPOSITIONS OF PARTIAL FUNCTIONS

In practice the function to be synthesized is often incompletely specified. Partial functions are also frequently encountered in constructing a sequence of decompositions where the initial function is total and many-valued. Such sequences are the basis of synthesis techniques based on decomposition. Decomposition techniques for partial functions are indispensable to developing efficient design techniques.

Hight [22] has considered the extension of Ashenhurst's criterion to partial two-valued functions. A total two-valued function has a simple disjunctive decomposition if, and only if, the corresponding partition matrix has at most two unique columns. In the case where the function is partial the question is how to interpret the don't-care conditions in the matrix.

Two columns are compatible if each row of the

matrix satisfies one of the following

i) one of the columns contains a don't-care in this row,

ii) the two columns are identical in this row.

Hight has shown that a partial two-valued function has a simple disjunctive decomposition if, and only if, the columns of the corresponding partition matrix can be partitioned into two sets of mutually compatible columns. α and g are determined in a similar manner to the one used by Waliuzzaman and Vranesic [82] in the case of total many-valued function.

Hight's result is a special case of a result due to Roth and Karp [59]. The latter characterizes a large class of decompositions of a many-valued function with or without don't-care conditions. The techniques to be developed in chapters 3 and 4 are based on this result.

Consider an arbitrary, possibly partial, many-valued function f(A). Let E, X, Y, Z be finite sets such that Z = R(f) and E = D(f) is a subset of the Cartesian product $X \times Y$. Suppose there exist functions g and α such that for all $e = \{x, y\} \in E$,

$$f[x, y] = g[\alpha[x], y] \qquad (6)$$

Note that $D(\alpha) = X$, $R(g) \supseteq R(f)$ and D(g) is a subset of the Cartesian product $R(\alpha) \times Y$.

E is the set of assignments to A for which f is defined. Expressing E in terms of the Cartesian product

$X \times Y$ in equivalent to partitioning A into two disjoint subsets A_λ and A_μ if we assume A consists of the variables of A_λ followed by the variables of A_μ. This is acceptable since the ordering of the arguments of a function is arbitrary. The decomposition in (6) can be expressed as

$$f(A) = g(\alpha(A_\lambda), A_\mu) \tag{7}$$

The range of α is not bounded as in the decompositions above. In practice we are concerned with how an r-valued function can be realized using a number of simpler r-valued functions. Thus in (7) when $|R(\alpha)| > |R(f)|$, α is interpreted as a t-tuple $\{\alpha_1, \alpha_2, \ldots, \alpha_t\}$ where $|R(\alpha_i)| \leq |R(f)|$, $1 \leq i \leq t$. Expression (7) then represents the decomposition

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

Given f, A_λ and A_μ we wish to determine:

i) given α under what conditions does g exist such that (7) holds?

ii) under what conditions do g and α exist such that (7) holds?

An answer to ii) will establish a criterion for the existence of a decomposition. An answer to i) will aid in determining g and α.

A compatibility relation over $X = D(\alpha)$ is the basis of these answers. $x_i, x_j \in X$ are <u>compatible</u> (denoted $x_i \sim x_j$) if, and only 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 and x_j are incompatible (denoted $x_i \not\sim x_j$).

Compatibility is reflexive ($x_i \sim x_i$), symetric ($x_i \sim x_j \Rightarrow x_j \sim x_i$) but not always transitive i.e. $x_i \sim x_j, x_j \sim x_k \not\Rightarrow x_i \sim x_k$. If f is a total function, compatibility is transitive and is an equivalence relation. To see this note that when f is total $E = X \times Y$. $x_i \sim x_j \Rightarrow f[x_i, y] = f[x_j, y]$ for all $y \in Y$ and $x_j \sim x_k \Rightarrow f[x_j, y] = f[x_k, y]$ for all $y \in Y$. Hence $x_i \sim x_j, x_j \sim x_k \Rightarrow f[x_i, y] = f[x_k, y]$ for all $y \in Y$ $\Rightarrow x_i \sim x_k$. For a partial function not every $\{x, y\}$ pair appears in E and we can have $x_i \sim x_j, x_j \sim x_k$ but $f[x_i, y_\ell] \neq f[x_k, y_\ell] \Rightarrow x_i \not\sim x_k$ so long as $f[x_j, y_\ell]$ is undefined.

Theorem 2.5. (Roth and Karp [59]) Given f and α there exists a g such that (6) holds if, and only if, for $x_i, x_j \in X$

$$\alpha[x_i] = \alpha[x_j] \Rightarrow x_i \sim x_j \tag{8a}$$

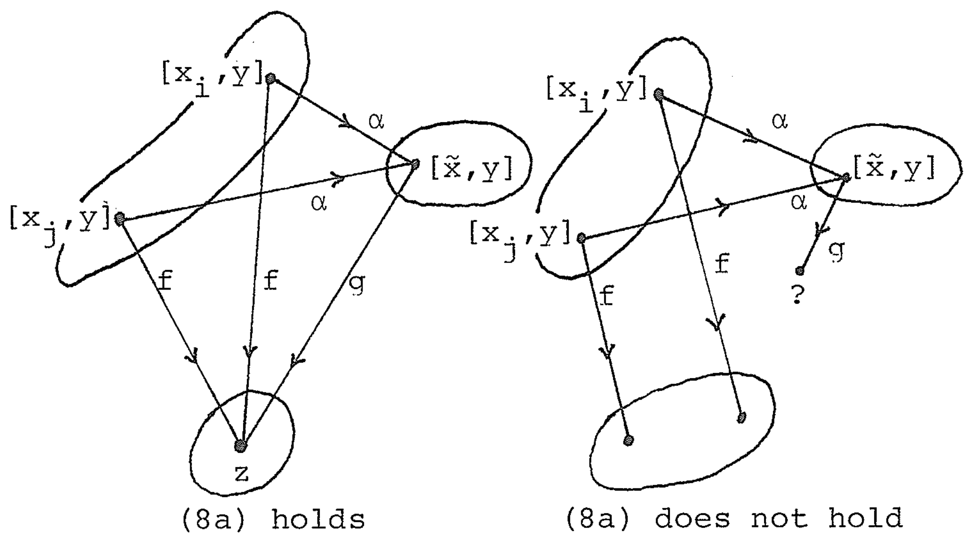
or equivalently

$$x_i \not\sim x_j \Rightarrow \alpha[x_i] \neq \alpha[x_j] \tag{8b}$$

Proof. Let $\tilde{x}$ denote an element in $R(\alpha)$ and let $\alpha^{-1}[\tilde{x}] = \{x_i, i = 1, 2, \ldots, q\}$ be the elements of X mapped onto $\tilde{x}$. If (8a) holds for each $y \in Y$, there exists a $z \in Z$ such that if $x_i \in \alpha^{-1}[\tilde{x}]$ and $\{x_i, y\} \in E$, $f[x_i, y] = z$. Thus we may define $g[\tilde{x}, y] = z$. The same process can be carried out for each element in $D(g)$.

Conversely if (8a) does not hold, there exists some $\{x_i, y\}, \{x_j, y\} \varepsilon E$ such that $f[x_i, y] \neq f[x_j, y]$ and $\alpha[x_i] = \alpha[x_j]$. Clearly there is no consistent assignment for $g[\tilde{x}, y]$. QED.

The situations in the proof are depicted below.



(8a) holds (8a) does not hold

FIGURE 2.2

Corollary 2.5.1. If k is the least integer such that $X = D(\alpha)$ can be partitioned into k classes of mutually compatible elements, then g and α exist so that (6) holds if, and only if, $|R(\alpha)| \geq k$.

Proof. Let k be the least integer such that X can be partitioned into k classes of mutually compatible elements denoted $C_1, C_2, \ldots, C_k$.

Suppose $|R(\alpha)| \geq k$. Choose α so that $\alpha[x_i] = \alpha[x_j]$ $\Leftrightarrow x_i, x_j \varepsilon C_\ell$ for some ℓ. Clearly $\alpha[x_i] = \alpha[x_j]$ $\Rightarrow x_i \sim x_j$ and by theorem 2.1, g exists.

Now suppose $|R(\alpha)| < k$. No matter how α is chosen, there is some x_i, x_j such that $\alpha[x_i] = \alpha[x_j]$ and $x_i \not\sim x_j$. By theorem 2.5 no g exists. QED.

As an illustration of these results consider the following function. Let $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3\}$ and $Z = \{z_1, z_2\}$. Let f be given by the table:

f	x_1	x_2	x_3
y_1	-	z_1	z_2
y_2	z_1	-	z_1
y_3	z_2	z_2	z_1

where '-' indicates a 'don't-care' condition. The set $E = \{\{x_1, y_2\}, \{x_1, y_3\}, \{x_2, y_1\}, \{x_2, y_3\}, \{x_3, y_1\}, \{x_3, y_2\}, \{x_3, y_3\}\}$.

$f[x_2, y_1] \neq f[x_3, y_1]$ and $x_2 \not\sim x_3$. Similarly $f[x_1, y_3] \neq f[x_3, y_3]$ and $x_1 \not\sim x_3$. For all y_ℓ such that $\{x_1, y_\ell\}, \{x_2, y_\ell\} \in E$, $f[x_1, y_\ell] = f[x_2, y_\ell]$ and thus $x_1 \sim x_2$. X can be partitioned into two subsets of mutually compatible elements $\{x_1, x_2\}$ and $\{x_3\}$. g and α exist if, and only if, $|R(\alpha)| \geq 2$.

Let $R(\alpha) = W = \{w_1, w_2\}$. g exists if, and only if, $\alpha(x_i) = \alpha(x_j) \Rightarrow x_i \sim x_j$. Choose $\alpha(x_1) = \alpha(x_2) = W_1$ and $\alpha(x_3) = W_2$. To find g simply substitute into $f[x_i, y_j] = g[\alpha[x_i], y_j]$. For example $g[w_1, y_1] = g[\alpha[x_2] = f[x_2, y_1] = z_1$. The following table defines g.

g	w_1	w_2
y_1	z_1	z_2
y_2	z_1	z_1
y_3	z_2	z_1

The choice of α is fairly free. We could have chosen $\alpha[x_1] = \alpha[x_2] = w_2$ and $\alpha[x_3] = w_1$. With a larger range many permutations are possible. We could increase the size of w and choose $\alpha(x_1] = w_1$, $\alpha[x_2] = w_2$ and $\alpha[x_3] = w_3$. This is permissible but of little use. In practice an α whose range is minimal is chosen.

Theorem 2.5 and corollary 2.5.1 can be used to determine the simple disjunctive decompositions of a many-valued function. Simple nondisjunctive decompositions could be determined in a manner similar to the one used by Curtis in the case of total two-valued functions. Karp [29] has suggested the following simpler approach.

A simple nondisjunctive decomposition is an expression of the form

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

where $A_\lambda \cup A_\mu = A$ and $A_\lambda \cap A_\mu \neq \phi$. $B = A_\lambda \cap A_\mu$ is termed the bound set. Let X represent the set of assignments assumed by A_λ and let Y represent the set of assignments assumed by $A_\mu - B$. The mutually compatible subsets of X are found as described above. In

the disjunctive case, α must map incompatible elements onto different images. In the nondisjunctive case this is only required if the elements agree in the coordinates corresponding to the bound variables.

To see this consider some $x_i \not\sim x_j$. From theorem 2.5 x_i and x_j must be mapped onto different images. The point is that the image is no longer simply α but α supplemented by the bound variables. If x_i and x_j disagree in a coordinate corresponding to a bound variable, the distinction of the images is immediate. If they agree in all coordinates corresponding to bound variables the distinction must be made by α. This approach can be applied to any total or partial many-valued function.

Extending the complex decomposition results to partial functions is a difficult problem as illustrated by the following example.

The function $f(a_1,a_2,a_3,a_4)$ given by

a_1	a_2	a_3	a_4	f	a_1	a_2	a_3	a_4	f
0	0	0	0	0	1	0	0	0	1
0	0	0	1	-	1	0	0	1	1
0	0	1	0	1	1	0	1	0	0
0	0	1	1	1	1	0	1	1	0
0	1	0	0	-	1	1	0	0	1
0	1	0	1	1	1	1	0	1	1
0	1	1	0	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	0

has decompositions

$$f(a_1, a_2, a_3, a_4) = g_1(\alpha_1(a_1, a_2), a_3, a_4)$$

and $f(a_1, a_2, a_3, a_4) = g_2(\alpha_2(a_3, a_4), a_1, a_2)$ as shown by the partition matrices

a_3a_4 \ a_1a_2	00	01	10	11
00	0	-	1	1
01	-	1	1	1
10	1	1	0	0
11	1	1	0	0

a_1a_2 \ a_3a_4	00	01	10	11
00	0	-	1	1
01	-	1	1	1
10	1	1	0	0
11	1	1	0	0

However there is no h such that

$$f(a_1, a_2, a_3, a_4) = h(\alpha_1(a_1, a_2), \alpha_2(a_3, a_4)).$$

To see this note that $\alpha_1(0,0) = \alpha_1(0,1)$ and $\alpha_2(0,0) = \alpha_2(0,1)$. It follows that $h(\alpha_1(0,0), \alpha_2(0,0)) = h(\alpha_1(0,1), \alpha_2(0,1))$. $f(0,0,0,0) \neq f(0,1,0,1)$ and there is thus no consistent assignment for h.

Hight [22] has observed that the existence of a simple decomposition of a partial function implies an assignment to the don't-care conditions of the function. This assignment is such that the resulting total function exhibits the same decomposition. Two simple decompositions are <u>compatible</u> if there is an assignment to the don't-cares of the function so that the resulting total function exhibits both decompositions. A partial function has a complex decomposition if, and only if, the simple decompositions comprising the complex decomposition are mutually compatible.

In the above example the decomposition

$$f(a_1, a_2, a_3, a_4) = g_1(\alpha_1(a_1, a_2), a_3, a_4)$$

implies $f(0,0,0,1) = 1$ and $f(0,1,0,0) = 0$.

The decomposition

$$f(a_1, a_2, a_3, a_4) = g_2(\alpha_2(a_3, a_4), a_1, a_2)$$

implies $f(0,0,0,1) = 0$ and $f(0,1,0,0) = 1$. There is thus no complex decomposition involving these decompositions.

There is no efficient technique for determining if two simple decompositions are compatible. The complex decomposition of partial functions is therefore not viable.

6. SYNTHESIS TECHNIQUES

The relationship between the decomposition properties of two-valued functions and the synthesis of switching circuits was recognized early in the history of switching theory [2], [53], [63], [65], [66], [67]. The first major study was presented by Ashenhurst [2]. His principal result is that a total two-valued function f has a unique disjunctive treelike decomposition (up to complementing of the α_j) such that the α_j are two-valued, and such that all simple disjunctive decompositions of f in terms of two-valued functions are exhibited. In other words no α_j has a simple disjunctive decomposition. An algorithm for the construction of this decomposition is given by Curtis [9].

As defined by Ashenhurst, a function f(A) is a disjunctive treelike decomposition of itself where $t = 0$, $A_1 = A$, $B_1 = \phi$ and $\alpha_1 = f$. It was shown by Shannon [65] that very few functions exhibit simple disjunctive decompositions. Ashenhurst's result is thus of little practical use on its own.

Roth and Wagner [60] devised a method for representing a total two-valued function as a disjunction of disjunctive treelike decompositions. All two-valued functions g such that

i) $g = 1 \Rightarrow f = 1$,

ii) g has a disjunctive treelike decomposition in terms of a given set of two-valued functions

are determined. The given functions correspond to available hardware devices. Once these g are found f is expressed as a disjunction of a minimal subset of them. The selection of this minimal subset is a process akin to prime implicant selection [55].

Roth and Wagner's method involves a significant amount of computation. More important however is the fact that generality is lost. Each function is realized as a disjunction of functions. There are many functions whose optimal realization is not of this form.

Curtis [9] has presented a method for the

synthesis of total two-valued functions which is based on his theory of proper complex decomposition. This method represents the broadest application of complex decomposition theory achieved to date. Unfortunately, since the theory is not easily extended to partial functions and is restricted to two-valued functions, this work sheds no light on the synthesis of partial many-valued functions.

Because of the restricted nature of the complex decomposition theory, a synthesis technique for partial many-valued functions must be based solely on simple decomposition. This is the approach which Roth and Karp [59] have adopted in their method for the synthesis of partial two-valued functions. For a given function f(A) all possible simple decompositions are determined. One is selected and its image g is found. The simple decompositions of g are found; one is selected; and the image is found. This process is repeated until a sequence of simple decompositions representing f(A) is obtained.

For example, the function $f(a_1, a_2, a_3, a_4)$ given in table 2.1 has a simple nondisjunctive decomposition

$$f(a_1, a_2, a_3, a_4) = g_1(\alpha_1(a_1, a_2), a_2, a_3, a_4)$$

where $\alpha_1(a_1, a_2) = \overline{a_1} \cdot a_2$. g_1 is given in table 2.2

g_1 has a simple disjunctive decomposition

$$g_1(\alpha_1, a_2, a_3, a_4) = g_2(\alpha_2(a_2, a_4), \alpha_1, a_3)$$

where $\alpha_2(a_2, a_4) = a_2 \cdot a_4$. g_2 is given by

TABLE 2.1

a_1	a_2	a_3	a_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	-
0	0	1	1	-
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

TABLE 2.2

α_1	a_2	a_3	a_4	g_1
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	-
1	0	0	1	-
1	0	1	0	-
1	0	1	1	-
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

α_1	a_3	α_2	g_2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

which has a simple disjunctive decomposition

$$g_2(\alpha_1, a_3, \alpha_2) = g_3(\alpha_3(a_3, \alpha_2), \alpha_1)$$

where $\alpha_3(a_3, \alpha_2) = a_3 \cdot \bar{\alpha}_2$. Finally g_3 is given by

α_1	α_3	g_3
0	0	0
0	1	1
1	0	1
1	1	1

which is a simple OR function. The complete decomposition is

$$f(a_1, a_2, a_3, a_4) = g_3(\alpha_3(a_3, \alpha_2(a_2, a_4)), \alpha_1(a_1, a_2))$$

where g_3, α_1, α_2 and α_3 are given above. This corresponds to the circuit given in the diagram below.

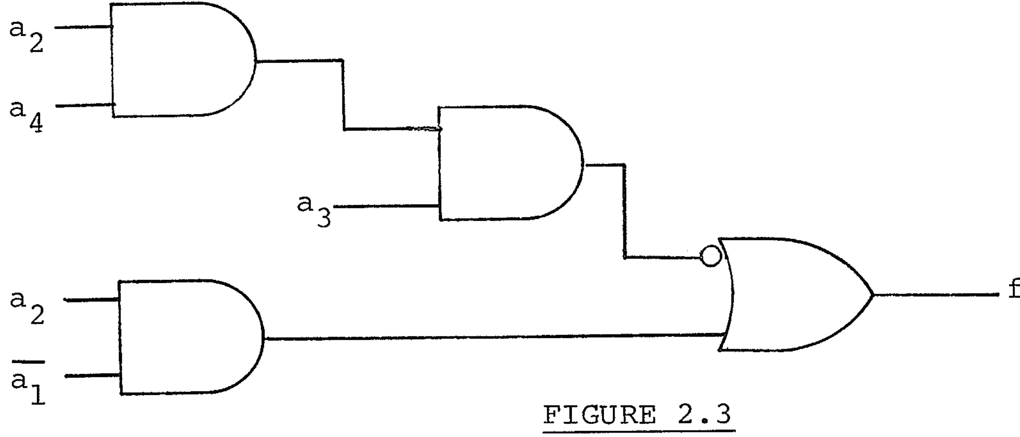


FIGURE 2.3

Roth and Karp's [59] algorithm potentially generates all possible decomposition sequences and hence all minimal circuits. This search is reduced by avoiding the generation of several sequences representing the same circuit. In addition, a cost bound is employed. If MIN is the cost of the minimal circuit which has been found to some point in the search, then any partial sequence whose cost exceeds MIN is discarded since any circuit found by extending this partial sequence will cost more than MIN. This use of a cost bound is extremely important since the termination of a partial sequence at a relatively early point prevents the formation of a number of uneconomical solutions.

Even with these refinements the search can be extremely time-consuming. For larger problems the search is so long that only a partial search can be

made and an approximately minimal solution accepted. Roth and Karp [59] estimated that a minimal circuit could be found for nearly any four variable problem in less than ten minutes on an IBM 7090. There has been no report on the validity of this estimate.

Karp *et al* [60] describe a computer program which implements Roth and Karp's algorithm. In this implementation, only decompositions where each α_i is a vertex function are considered. A *vertex function* is one which assumes the value 0 or 1 for one input condition and the other value for all other input conditions. The restriction to vertex decomposition allows the use of a decomposition test which is far more efficient than the test required in the general case. A number of heuristics are described which bias the program toward the early generation of economical circuits. In this way, nearly minimal circuits are obtained with a very short search.

Roth and Karp's algorithm [59] and the implementation due to Karp *et al* [60] are the starting point for the synthesis procedures developed in chapters 5 and 6 and they are discussed there in more detail. One very interesting observation is that although both the algorithm and the computer program are described as permitting α_i with any number of arguments, all the examples involve only two-input gates with the exception of a single three-input gate.

THE UNIVERSITY OF MANITOBA LIBRARIES

7. REMARKS

The method used to identify decompositions is the most important component in a synthesis algorithm employing decomposition. This is the most time consuming process and its efficiency is critical.

Partition matrices are a useful pedagogical tool. They are easily understood and are invaluable illustrations of decomposition. For two-valued functions with up to four or five variables the construction and analysis of partition matrices is reasonable for hand computation. As the number of arguments increases the process becomes more and more tedious and prone to error. For many-valued functions hand solution is impractical.

A computer can be used but this solution is not attractive for two reasons. First, the function must be given as a truth table. A table describing an n-argument r-valued function has r^n entries. For values as small as $r = 5$, $n = 4$ this number is quite large ($5^4 = 625$). Either a large amount of data must be prepared or the computer must be used to convert a more compact specification to a truth table.

The second problem is the work required to manipulate matrices of this size. For each A_λ, A_μ partition the truth table must be converted to the corresponding partition matrix and the columns examined to determine γ or sets of mutually compatible columns. The analysis of the columns is a form of pattern recognition. It is

an easy visual task but on the computer requires exhaustive comparison. These problems are not discussed in the literature and after considerable work the author has found no efficient methods.

The application of Roth and Karp's results (theorem 2.5 and corollary 2.5.1) is more suited to the computer. The function is not required in truth table form and is in fact accepted in a very compact notation which is easily handled by the computer. A major advantage is that the specification of the function is the same for each $\lambda\mu$-partition. This is in contrast to the complexity of constructing the partition matrices. The implementation of these results is discussed in detail in the next chapter. It is the author's experience that this approach is far more efficient than using partition matrices.

A synthesis technique must be able to handle partial functions. This is the case even if the given function is total since for certain decompositions there is no guarantee that g is total. Consider the decompositions characterized by theorem 2.5. These take the form

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

where $A_\lambda \cup A_\mu = A$, $A_\lambda \cap A_\mu = \phi$ and where each α_i, $1 \leq i \leq t$ assumes no more values than f. It is possible for $\alpha_1, \alpha_2, \ldots, \alpha_t$ to assume a subset of the possible

configurations and hence for g to be partial.

Complex decomposition does not extend to partial many-valued functions. The synthesis technique developed in chapter 5 therefore employs the decompositions characterized by theorem 2.5 in an approach based on the method suggested by Roth and Karp [59] i.e. the construction of circuits by forming a sequence of decompositions. It will also be found that this approach results in an efficient two-valued synthesis technique. In particular, an algorithm for the synthesis of multiple-output two-valued switching circuits will be presented. This problem has previously been computationally too complex to solve using decomposition techniques.

CHAPTER 3

Decomposition of Many-Valued Functions

1. INTRODUCTION

The application of decomposition techniques to the synthesis of switching circuits involves three steps; the identification of decompositions; the assignment of the functions in the decompositions; the construction of a sequence of decompositions representing the desired switching circuit. In this chapter the first two steps are considered for the case of many-valued functions. Two-valued functions are considered in chapter 4. The construction of sequences of decompositions representing switching circuits is discussed in chapters 5 and 6.

Karp [29] and Waliuzzaman and Vranesic [82] have considered the identification of many-valued decompositions. These results were examined in chapter 2. Karp's results are concerned with the theoretical basis for identifying various forms of complex decompositions. The practical implementation of these results and the assignment problem were not considered by Karp. Waliuzzaman and Vranesic extended the partition matrices of Ashenhurst [2] to many values. This work was restricted to total functions and as we have seen in chapter 2 is not easily adapted to partial functions.

The following discussion considers the identification and assignment of decompositions of the form

$$f(A) = g(\alpha_1(A_\lambda), \alpha_2(A_\lambda), \ldots, \alpha_t(A_\lambda), A_\mu) \qquad (1)$$

where $A_\lambda \cap A_\mu = \phi$, $A_\lambda \cup A_\mu = A$ and each α_i assumes no more values than f. f is either total or partial. This is important since even if the initial function is total, partial functions are often encountered in constructing a sequence of decompositions representing a switching circuit. The decompositions in (1) are a subset of the decompositions characterized by theorem 2.5 and corollary 2.5.1. The emphasis here is the practical implementation of these results.

We shall find that implementing these results in their full generality is an extremely complex problem. In particular, there is no efficient algorithm for assigning the α_i. The case where $|A_\lambda| = 2$ and $1 \le t \le 2$ is considered in detail. Both the identification and assignment problems are much simpler in this case and reasonably efficient algorithms are developed. Fortunately, this class of decompositions is very useful in the practical design problem.

Consider an r-valued function $f(a_1, a_2, \ldots, a_n)$ possibly with 'don't-care' conditions. Let E be the set of input configurations for which f is defined. Each element of E is an n-tuple of symbols taken from $\{0, 1, \ldots, r-1\}$. Let Z, a subset of $\{0, 1, \ldots, r-1\}$ be the values assumed by f.

A decomposition of the form (1) involves a partitioning of the variables $a_1, a_2, \ldots, a_n$ into two subsets A_λ and A_μ. Corresponding to this partition is a partitioning of the elements of E written $e_i = (x,y)$ where x is an s-tuple and y is an (n-s)-tuple. The elements of x are the elements of e_i corresponding to variables in A_λ. The elements of y are the elements of e_i corresponding to variables in A_μ. Let X denote the set of all x found in this manner. Let Y denote the set of all y found in this manner. By applying an appropriate permutation to the coordinates of the elements of E, we can arrange to have E a subset of $X \times Y$ (the Cartesian product of X and Y). If f is total, $E = X \times Y$.

For example, consider $f(a_1, a_2, a_3)$ given by the following truth table:

a_1	a_2	a_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	2	1
0	2	1	1
0	2	2	2

a_1	a_2	a_3	f
1	0	0	0
1	0	1	1
1	1	0	0
2	1	2	2
2	2	1	2
2	2	2	2

f is a partial three-valued function. E = {000, 001, 010, 012, 021, 022, 100, 101, 110, 212, 221, 222}. Let $A_\lambda = \{a_1, a_3\}$ and $A_\mu = \{a_2\}$. This partitions the elements of E. For example, 000 is partitioned into x=00

and y=0; 021 is partitioned into x=01 and y=2. If this is done for each element in E we find X={00, 01, 02, 10, 11, 21, 22} and Y={0, 1, 2}. If the above table is rewritten with the arguments ordered a_1, a_3, a_2, E is a subset of X × Y.

We can rewrite (1) as

$$f[x,y] = g[\alpha[x], y] \qquad (2)$$

where α denotes the t-tuple $\{\alpha_1, \alpha_2, \ldots, \alpha_t\}$. Recall the following definition, theorem and corollary.

Definition. x_i, $x_j \in X$ are compatible (denoted $x_i \sim x_j$) if, and only 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 and x_j are incompatible (denoted $x_i \not\sim x_j$).

Theorem 2.5. (Roth and Karp [59]) Given f and α, there exists a g such that (2) holds 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

$$x_i \not\sim x_j \Rightarrow \alpha[x_i] \neq \alpha[x_j]$$

Corollary 2.5.1. If k is the least integer such that X can be partitioned into k classes of mutually compatible elements, then g and α exist so that (2) holds if, and only if, there are at least k elements in the range of α.

2. A REPRESENTATION FOR MANY-VALUED FUNCTIONS

There are a number of constraints on the notation used to represent a many-valued function. It must be easily understood and should create as little overhead as possible in preparing a problem for solution on the computer. The notation should be easy to represent and manipulate on the computer and should be as concise as possible. The latter constraint is concerned both with efficient utilization of the computer memory and with reducing the computation required to perform operations on functions. Determining the compatible and incompatible elements of X and the images of decompositions are the two steps which refer to the function being considered. The efficiency in performing these operations is a major consideration in choosing a notation.

An r-valued function $f(a_1, a_2, \ldots, a_n)$ will be represented by a matrix F whose elements are subsets of $\{0, 1, \ldots, r-1\}$. We shall write F_i to denote the ith row of F and F_{ij} to denote the element in the ith row and j^{th} column. The matrix has n+1 columns labeled $a_1, a_2, \ldots, a_n, f$.

Each row F_i represents a set of assignments to $\{a_1, a_2, \ldots, a_n\}$ i.e. a subset of E. For each assignment the value of a_j is in F_{ij}. The complete set of assignments is found by assigning values to the a_j in all possible combinations. The function repre-

sented by F assumes the same value for all the assignments represented by F_i. This value is given by $F_{i(n+1)}$.

Consider the function given by table 3.1. f assumes the value 0 for each of the following input conditions:

000	020	200	220
001	021	201	221
002	022	202	222

f is thus 0 whenever a_1 and a_2 are assigned values from {0, 2} and a_3 is assigned a value from {0, 1, 2}. In matrix notation these twelve conditions are specified by the single row

a_1	a_2	a_3	f
{0,2}	{0,2}	{0,1,2}	{0}.

If $a_1 = 0$, $a_2 = 1$ and a_3 is assigned a value from {0, 1}, f assumes the value 1. This is written as

a_1	a_2	a_3	f
{0}	{1}	{0,1}	{1} .

A complete matrix representation of f is given in table 3.2. The brackets have been omitted for clarity.

This matrix notation is particularly suited to the computer. Each of the sets is represented by a bit string. A 1 in the ith position of this string indicates the set being represented contains the ith value while a 0 in the ith position indicates it does not contain the ith value. In our implementation on

TABLE 3.1

a_1	a_2	a_3	f	a_1	a_2	a_3	f	a_1	a_2	a_3	f
0	0	0	0	1	0	0	1	2	0	0	0
0	0	1	0	1	0	1	2	2	0	1	0
0	0	2	0	1	0	2	0	2	0	2	0
0	1	0	1	1	1	0	2	2	1	0	2
0	1	1	1	1	1	1	2	2	1	1	1
0	1	2	0	1	1	2	0	2	1	2	0
0	2	0	0	1	2	0	1	2	2	0	0
0	2	1	0	1	2	1	1	2	2	1	0
0	2	2	0	1	2	2	0	2	2	2	0

TABLE 3.2

	a_1	a_2	a_3	f
1 -	02	02	012	0
2 -	0	1	01	1
3 -	1	1	01	2
4 -	012	1	2	0
5 -	12	1	0	2
6 -	02	1	1	1
7 -	1	2	01	1
8 -	1	02	2	0
9 -	1	01	1	2
10 -	1	02	0	1

an IBM 370/158 computer, functions in up to eight values are considered and a single byte (8 bits) is used to represent each set in the matrix. Determining the incompatible elements of X and the images of decompositions requires manipulation of these sets which on the computer is accomplished using simple bit string operations. The techniques used are discussed in detail in chapter 5.

For the designer, construction of the matrix representation is straightforward. Design problems are typically stated as descriptions of how the output should behave according to conditions on the input. For example, a problem might specify the output is 0 whenever there are two or more 0's in the input assignment, otherwise the output assumes the maximum input value. These sorts of descriptions are easily transformed into matrices. For four arguments and three values the function required in the above problem is given by the following pair of tables:

a_1	a_2	a_3	a_4	f
0	0	012	012	0
0	012	0	012	0
0	012	012	0	0
012	0	0	012	0
012	0	012	0	0
012	012	0	0	0

a_1	a_2	a_3	a_4	f
012	12	12	12	2
12	012	12	12	2
12	12	012	12	2
12	12	12	012	2

The rows in the first table represent the six ways of assigning 0 to two arguments. In these cases the values of the other two arguments may be 012. The rows in the second table each assign at most one 0 to an argument. It is easily verified that these tables represent all possible input assignments.

3. THE DETERMINATION OF k

Given f and a particular $\lambda\mu$-partition the first steps in determining if a decomposition of the form (1) exists is to determine the compatible and incompatible pairs of X and from these to extract the partition of X and value k described in corollary 2.5.1. We shall examine the problem of determining the compatible and incompatible elements of X when f is given in the matrix notation.

Consider F_i the ith row of F. The first n columns of F_i represent a subset of E. Since E is a subset of $X \times Y$, F_i defines a subset of X and a subset of Y.

The subset of X defined by F_i is the set of input assignments defined by F_{iq_1}, F_{iq_2}, ..., F_{iq_s} where each $a_{q_t} \varepsilon A_\lambda$. The subset of Y defined by F_i is the set of input assignments defined by F_{iq_1}, F_{iq_2}, ..., $F_{iq_{n-s}}$ where each $a_{q_t} \varepsilon A_\mu$. The following algorithm is used to find the compatible and incompatible elements of X.

Algorithm 3.1

To begin all pairs of elements of X are compatible. For each pair of rows F_i, F_j in F:

1. if $F_{i(n+1)} = F_{j(n+1)}$ ignore this pair;
2. find Y_i the subset of Y defined by F_i and Y_j the subset of Y defined by F_j;
3. if $Y_i \cap Y_j = \phi$ ignore this pair;
4. find X_i the subset of X defined by F_i and X_j the subset of X defined by F_j;
5. record that each element in X_i is incompatible with each element in X_j and go to the next pair.

This algorithm is simply an iterative application of the definition of compatibility. Examining all pairs ensures that all e_i, $e_j \varepsilon E$ such that $f[e_i] \neq f[e_j]$ are compared. Step 1 avoids comparing two e_i, e_j when $f[e_i] = f[e_j]$. Consider the rows F_i, F_j and assume $F_{i(n+1)} = F_{j(n+1)}$. If $Y_i \cap Y_j \neq \phi$ then there is a $Y \varepsilon Y_i$ such that $y \varepsilon Y_j$ and hence by definition each $x_p \varepsilon X_i$ is incompatible with each $x_q \varepsilon X_j$ since (x_p, Y),

$(x_q, y) \varepsilon E$ and $f[x_p, y] \neq f[x_q, y]$. If $Y_i \cap Y_j = \phi$ we proceed to the next pair since no assignment specified by F_i has a common y part with any assignment specified by F_j. This process identifies incompatible pairs of elements of X. All other pairs are by definition compatible.

Consider the function defined by

	a_1	a_2	a_3	a_4	f
1-	0	0	0	1	0
2-	0	1	0	0	0
3-	1	01	0	1	0
4-	1	01	1	01	1
5-	1	01	0	0	1
6-	0	01	1	01	1
7-	0	0	0	0	1
8-	0	1	01	1	1

Let $A_\lambda = \{a_1, a_2\}$ and $A_\mu = \{a_3, a_4\}$. The application of algorithm 3.1 is traced in table 3.3. To simplify this table pairs of rows agreeing in the value assigned to f have been omitted. The compatible and incompatible pairs of elements of X are

$00 \not\sim 01$ $\qquad$ $00 \sim 10$

$01 \not\sim 10$ $\qquad$ $00 \sim 11$

$01 \not\sim 11$ $\qquad$ $10 \sim 11$

TABLE 3.3

i	j	Y_i	Y_j	$Y_i \cap Y_j$	X_i	X_j	RESULT
1	4	{01}	{10,11}	ϕ			ignore this pair
1	5	{01}	{00}	ϕ			ignore this pair
1	6	{01}	{10,11}	ϕ			ignore this pair
1	7	{01}	{00}	ϕ			ignore this pair
1	8	{01}	{00,01}	{01}	{00}	{01}	00 $\not\sim$ 01
2	4	{00}	{10,11}	ϕ			ignore this pair
2	5	{00}	{00}	{00}	{01}	{10,11}	01 $\not\sim$ 10, 01 $\not\sim$ 11
2	6	{00}	{10,11}	ϕ			ignore this pair
2	7	{00}	{00}	{00}	{01}	{00}	00 $\not\sim$ 01
2	8	{00}	{01,11}	ϕ			ignore this pair
3	4	{01}	{00}	ϕ			ignore this pair
3	5	{01}	{00}	ϕ			ignore this pair
3	6	{01}	{10,11}	ϕ			ignore this pair
3	7	{01}	{00}	ϕ			ignore this pair
3	8	{01}	{01,11}	{01}	{10,11}	{01}	01 $\not\sim$ 10, 01 $\not\sim$ 11

For all other i,j pairs, f assumes the same value for both rows.

This algorithm is straightforward but rather tedious for hand execution. However, because of the efficient representation of the matrix of sets as a matrix of bit strings the algorithm can be implemented very efficiently on the computer.

Once the compatible and incompatible elements of X are known, the partition and value k described in corollary 2.5.1 must be found.

If f is a total function, compatibility is transitive and partitions X into equivalence classes. In this case the answer is immediate. In general, f may be partial, compatibility is no longer transitive and the problem is much more difficult. There is no efficient method for determining if a function is total. The general method must be applied in all cases.

Roth and Karp [59] have described a method for determining a minimal paritition of X when f is two-valued. This method is analogous to a state minimization technique due to Miller [41]. The algorithm depends solely on the compatibility relation and not on the fact that f is two-valued. It is thus applicable to many-valued functions.

A <u>maximal compatible subset</u> of X is one such that each pair of elements in the subset is compatible, and no element can be added without violating this property. Roth and Karp's algorithm determines a minimal number of these sets whose union is X. This minimal number

is the value k in corollary 2.5.1. The required partition of X is found by removing common elements from all but one of the chosen maximal compatible subsets.

Roth and Karp's algorithm is not considered in detail since we shall show there are other problems which make this approach to identifying decompositions of many-valued functions impossible.

4. THE CHOICE OF α

α must be chosen so that it satisfies theorem 2.5 i.e. if α maps two elements into the same image they must be compatible. A straightforward method of identifying a valid α is to determine a partition of X using Roth and Karp's algorithm and to then assign each set in this partition an element in the range of α. α is defined so that the elements in each set of the partition are all mapped into the image element associated with that set. This α satisfies theorem 2.5 since the elements mapped into a common element are mutually compatible. An α chosen in this manner has a minimal range.

The drawback to this approach is the number of choices. There are often several minimal partitions of X and for each partition there are many possible α. α represents a t-tuple of functions $\{\alpha_1, \alpha_2, \ldots, \alpha_t\}$ where each α_i assumes no more values than f. If f is r-valued, the minimum value of t so that theorem 2.5

holds is $r^{t-1} < k \le r^t$ where k is the number of sets in the partition of X. For the minimum value of t we term every α satisfying theorem 2.5 an assignment.

Theorem 3.1. Given maximal compatible subsets M_1, M_2, ..., M_k whose union is X, the number of assignments of α when f is r-valued is

$$\frac{(r^t)!}{t!} \cdot \sum_{j=k}^{r^t} \frac{n(j)}{(r^t-j)!}$$

where $r^{t-1} < k \le r^t$ and n(j) is the number of ways of partitioning X into j nonempty sets, each being a subset of some M_i.

Proof. Given a partition of X into j nonempty sets satisfying the above condition, the number of assignments such that two elements of X yield the same value for α if, and only if, they are in the same set of the partition is

$$\frac{(r^t)!}{t!(r^t-j)!}$$

i.e. $\frac{(r^t)!}{(r^t-j)!}$ the number of ways of assigning each set in the partition a unique element in the range of α, divided by t! since any two orderings of $\{\alpha_1, \alpha_2, \ldots, \alpha_t\}$ are distinquishable.

The total number of assignments is found by summing $\frac{(r^t)!}{t!(r^t-j)!}$ over all possible partitions of which there are $\sum_{j=k}^{r^t} n(j)$. QED.

This theorem is a generalization of a result due to Karp [29] who considered the case of r=2. The value of n(j) is given by $\Sigma(\prod_{i-1}^{k} S(n_i, m_i))$ where S(n, m) denotes the Stirling numbers of the second kind [61], n_i is the number of elements in M_i, and the summation has a term for each k-tuple of positive integers $(m_1, m_2, \ldots, m_k)$ such that $m_1 + m_2 + m_3 + \ldots + m_k = j$.

To appreciate the size of the problem consider the following example given by Curtis [10]. f is two-valued and the sets M_i are

M_1 = {0001, 1000, 1001, 1101}, M_2 = {1010, 1100},
M_3 = {0111, 1011, 1110, 1111}, M_4 = {0000, 0011, 0110},
M_5 = {0010, 0100}, M_6 = {0101}.

Here r = 2, k = 6, t = 3, n(6) = 1, n(7) = 19, n(8) = 139. The number of possible assignments is 1,065,120. This value becomes exponentially larger as r increases i.e. if f is many-valued function.

Of the possible assignments, the most interesting are those where the range of α is minimal. This value is given by

$$\frac{(r^t)!\ n(k)}{t!(r^t-k)!}$$

where k is the number of sets in the partition. For the example above there are

$$\frac{(2^3)!\ n(6)}{3!(2^3-6)!} = 672$$

minimal assignments. This is more reasonable than

the total number of assignments but is still quite large.

In many instances, the minimum set of mutually compatible elements whose union is X is not unique. This of course compounds the problem by increasing the number of possible assignments.

In the practical design problem we wish to choose α so that $\alpha_1, \alpha_2, \ldots, \alpha_t$ have as simple hardware realizations as possible. Some subset of the possible assignments can then be considered 'best' assignments since they have minimal hardware cost. An algorithm for choosing a 'best' assignment would be extremely valuable.

Choosing a 'best' assignment is a form of local optimization. It does not ensure that the overall circuit will be minimal but should lead in this direction. The large number of possible assignments makes it impossible to search all possible decomposition sequences. Karp [29] has studied the choice of a 'best' assignment for the case where f is two-valued. To the author's knowledge this is the only study of this area.

Karp states that the problem of choosing an assignment to minimize the costs of realizing $\alpha_1, \alpha_2, \ldots, \alpha_t$ and g appears to be far too difficult to solve completely. The above discussion certainly supports this view. As a partial solution Karp considers a procedure which

tends to reduce this cost by choosing the α_i in so far as possible from various classes of relatively simple functions. The classes considered include functions which are independent of certain of their arguments, functions possessing certain simple decompositions, symmetric functions, unate functions and threshold functions. The process involves two steps: the determination of assignable functions belonging to these classes and the selection of assignments using these functions.

Karp's paper [29] is divided into two sections: a discussion of the theory of decomposition and a discussion of the application of this theory to circuit synthesis. The first section is quite general and considers both partial and many-valued functions. The second section which includes the results on the assignment of the α_i, is restricted to total two-valued functions.

The partitioning of X into sets of mutually compatible elements is fundamental to Karp's assignment results. When the function being considered is total, this partitioning is trivial since the compatibility relation partitions X into equivalence classes. In this case, a minimal number of mutually compatible subsets of X whose union is X must be found. This minimal partitioning is seldom unique. The amount of computation is greatly compounded since not only must

all the minimal partitions be determined but the assignments derived from each must be examined. Karp indicates a direction for the extension of his results to partial two-valued functions but does not pursue the area in any detail.

Karp does not consider the assignment of the α_i for many-valued functions. When the function being considered is total this extension should not be difficult. The range of each α_i is of course larger and this increases the computation by increasing the number of possibilities. The analysis of each possibility should remain essentially the same. The domain of the α_i presents no problem since these are treated as elements of the set X independent of the base of the function.

The extension of Karp's assignment results to partial or many-valued functions will not be attempted in this thesis. These extensions are important to developing the understanding of the composition of switching functions and are in that context important areas of research. The objective of this thesis is, however, to develop efficient synthesis techniques which as we mentioned earlier must be applicable to partial functions. It is this author's opinion that the computation resulting from extending Karp's results to partial functions would be prohibitive in practice. This belief is based on the fact that the determination

of the minimal partitions of X for a partial function is known to be an extremely complex problem which grows exponentially with the size of X. Computationally equivalent problems have been pursued at great length in the literature with little success in terms of an efficient algorithm. Two examples are prime implicant selection and the determination of the cliques of a graph.

Other approaches to decomposition are possible. For example, in some applications it may be reasonable to choose the α_i from a specific set of functions, assume a decomposition exists and then proceed to construct the image of this decomposition until either the image is completely defined or until a contradiction is found in which case the decomposition does not in fact exist. Such an approach might be taken if the designer were interested in whether or not a function can be realized using certain primitives such as threshold gates.

If the function to be realized is itself in a particularly tractable class of functions such as linear or symmetric functions, the decomposition problem may be simpler. Shannon [65] has considered the case where the function to be realized is symmetric and two-valued. Mukhopadhyay [85] has presented an extension of Shannon's results to three-valued symmetric functions. In these cases an analytical solution is

arrived at quite easily. These solutions are a realization of the function by a circuit with a predefined form. It is not clear to what degree these solutions are optimal.

In the discussion below, an alternative approach is taken. The emphasis is on developing an efficient synthesis technique applicable to both partial and many-valued functions. The comparison of this approach to those outlined above is beyond the scope of this thesis. It should however be pursued in future research.

5. TWO-PLACE DECOMPOSITION

When a general problem is unsolvable it is often rewarding to examine special cases of the problem. For the decomposition of many-valued functions we shall consider expressions of the form (1) where s=2 and $1 \le t \le 2$. These are termed two-place decompositions since α_1 and α_2 are at most two-place functions. This is the simplest nontrivial subclass of the decompositions in (1). Efficient algorithms for identifying two-place decompositions and for assigning α_1 and α_2 will be developed.

The two-place decomposition of two-valued functions has been considered by Barnard and Holman [3] and by Muzio and the author [36], [39], [40]. This case is considered in detail in the next chapter where very simple and efficient computer oriented methods are developed. Muzio and the author [38], [45] have

considered the two-place decomposition of three-valued functions. The techniques described below are a continuation of these results.

Consider an r-valued function $f(a_1, a_2, \ldots, a_n)$. A <u>two-place decomposition</u> is an expression of the form

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_1(a_i, a_j), \alpha_2(a_i, a_j), a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) \quad (3)$$

where α_1 and α_2 assume no more values than f. The decomposition is <u>nontrivial</u> if g is defined for fewer input conditions than f. An equivalent condition is that the mapping α denoting the 2-tuple (α_1, α_2) must have a range smaller than its domain.

A many-valued function $f(a_1, a_2, \ldots, a_n)$ is <u>degenerate</u> if there exists a function g such that

$$f(a_1, a_2, \ldots, a_n) = g(b_1, b_2, \ldots, b_m) \quad (4)$$

where $b_k \varepsilon \{a_1, a_2, \ldots, a_n\}$, $1 \leq k \leq m$ and $m < n$. The <u>degree</u> of f, denoted $\delta(f)$, is the smallest m such that (4) holds. If $\delta(f) = 0$, f is a constant.

If f is nondegenerate there are four types of nontrivial two-place decompositions according to the degrees of α_1 and α_2. Since the labelling of the inputs of g is arbitrary we choose α_1 and α_2 so that $\delta(\alpha_1) \leq \delta(\alpha_2)$. The four types of decomposition are as follows:

TYPE	$\delta(\alpha_1)$	$\delta(\alpha_2)$
I	0	2
II	1	1
III	1	2
IV	2	2

For a given $f(a_1, a_2, \ldots, a_n)$ and a_i, a_j pair there are often a number of nontrivial decompositions of varying types. We shall develop techniques for determining each of these decompositions. The problem of choosing which of these decompositions is of most practical use will be considered later.

The methods to be developed are based on graph theory. Harary [20] has described graph theory as a mathematical model for any system containing a binary relation. In our problem the binary relation is incompatibility. A few definitions are required to define the model.

A graph G consists of a finite nonempty set N(G) of p nodes together with a finite set L(G) of unordered pairs of distinct nodes of G. Each pair $\ell = \{n_1, n_2\}$ of nodes in L(G) is a line of G, and ℓ is said to join n_1 and n_2. We write $\ell = n_1 n_2$ and say that n_1 and n_2 are adjacent; node n_1 and line ℓ are termed incident as are n_2 and ℓ. If two distinct lines are incident with a common node they are adjacent lines. A graph is usually represented by a diagram such as the one below.

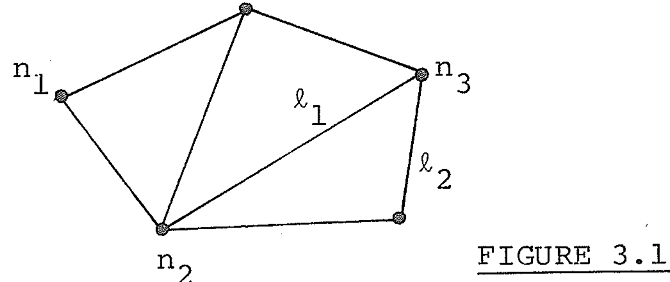


FIGURE 3.1

Here G has five nodes and seven lines. Nodes n_1 and n_2 are adjacent; nodes n_1 and n_3 are not. The line ℓ_1 joins n_2 and n_3 so ℓ_1 and n_2 and ℓ_1 and n_3 are incident. ℓ_1 and ℓ_2 are adjacent lines since they are incident with n_3.

Suppose for a given r-valued function $f(a_1, a_2, \ldots, a_n)$ and $a_i a_j$ pair the compatible and incompatible pairs of X have been determined (recall that X is the set of input configurations assumed by $\{a_i, a_j\}$). These relations can be represented by a graph G where N(G) has a unique node for each element of X. The nodes are distinquished by labeling each of them with a unique $x_k \in X$. The set L(G) is such that a pair of nodes are adjacent if, and only if, these labels, x_ℓ, x_m, are incompatible. For example given $X = \{00,01,02,10,11,12,20,21,22\}$ and the incompatibilities

$00 \not\sim 01$	$02 \not\sim 12$	$11 \not\sim 22$
$00 \not\sim 12$	$10 \not\sim 20$	$20 \not\sim 21$
$01 \not\sim 11$	$10 \not\sim 22$	$21 \not\sim 22$

with all other pairs of X being compatible, the corresponding graph is given by figure 3.2.

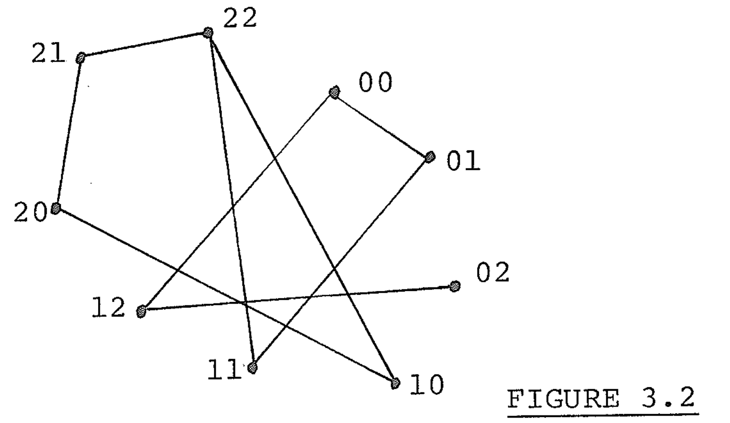


FIGURE 3.2

Graphs constructed in this manner are termed incompatibility graphs.

A colouring of a graph is an assignment of colours to the nodes so that no two adjacent nodes have the same colour. An n-colouring of a graph uses n colours. We shall write $\chi(G)$ to denote the minimum value of n such that G has an n-colouring. $\chi(G)$ is the chromatic number of G.

Graph colouring has been extensively studied and several algorithms for colouring graphs have been presented. This work is of interest in the study of two-place decomposition since the following results show the connection between α_1, α_2 and colourings of incompatibility graphs.

Given a nondegenerate r-valued function $f(a_1, a_2, \ldots, a_n)$ and an $a_i a_j$ pair, let G be the incompatibility graph representing the incompatible elements of X. p is the number of nodes in G i.e. the number of elements in X. Note $\chi(G) \leq p$.

Theorem 3.2. There exists a nontrivial two-place decomposition

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_1(a_i, a_j), \alpha_2(a_i, a_j), a_3, \ldots$$
$$\ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n)$$

if, and only if, $\chi(G) < p$.

Proof. Suppose $\chi(G) < p$. Consider a $\chi(G)$ - colouring of G and let c_t, $0 \le t < \chi(G)$, denote the colours used. Define α so that for all $x_t \in X$, $\alpha[x_t] = s$ where for the colouring being considered the node labeled x_t is coloured c_s. Since no adjacent nodes have the same colour, $\alpha[x_t] = \alpha[x_s] \Rightarrow x_t \sim x_s$ and by theorem 2.5 a decomposition exists. The decomposition is nontrivial since the range of α is smaller than its domain.

Suppose $\chi(G) = p$. It follows that each pair of nodes of G are adjacent and thus $x_t \not\sim x_s$ for all x_t, $x_s \in X$. By theorem 2.5 a decomposition exists if, and only if, $x_t \not\sim x_s \Rightarrow \alpha[x_t] \neq \alpha[x_s]$ hence α has p elements in its range and if $\chi(G) = p$ the corresponding decompositions are trivial. QED.

The above theorem provides a necessary and sufficient condition for the existence of a nontrivial two-place decomposition. Given that such a decomposition exists the following results provide conditions on its type.

6. TYPE I DECOMPOSITION

Theorem 3.3. There exists a nontrivial type I decomposition if, and only if, $\chi(G) \le r$, $\chi(G) < p$.

Proof. Suppose $\chi(G) \leq r$ and $\chi(G) < p$. By theorem 3.1 a nontrivial two-place decomposition exists. To show the decomposition is type I consider a $\chi(G)$ - colouring of G and let c_t, $0 \leq t < \chi(G)$, denote the colours. Define $\alpha_1 = 0$ and for each $x_t \in X$, define $\alpha_2[x_t] = s$ where the node labeled $x_t \in G$ is coloured c_s in the colouring being considered. α_2 assumes no more than r values since $\chi(G) \leq r$ hence the decomposition is type I.

Suppose a nontrivial type I decomposition exists. It follows from theorem 3.2 that $\chi(G) < p$. Suppose α_2 assumes k values. By definition $k \leq r$. Choose k colours c_t, $0 \leq t < k$. For each $x_t \in X$, colour the node labeled x_t the colour c_s where $\alpha_2[x_t] = s$. This defines a k-colouring of G since $\alpha_2[x_t] = \alpha_2[x_2] \Rightarrow x_t \sim x_s$ and the nodes labeled x_s and x_t are not adjacent in G. Since $k \leq r$, $\chi(G) \leq r$. QED.

7. TYPE II DECOMPOSITIONS

The only two-valued one-place functions are the constants, the variable and the inverse of the variable. If f is a nondegenerate two-valued function, there are thus no nontrivial type II decompositions. However, for more than two values there are many interesting one-place functions. There are $r^r - r - (r!)$ r-valued one-place functions whose ranges are smaller than their domains.

For r = 3 there are 18; for r = 4 there are 228. These are the functions to be considered in type II decompositions. To the author's knowledge no study of decompositions involving one-place functions has appeared in the literature.

Since f is nondegenerate, α_1 and α_2 must be functions of different variables. A type II two-place decomposition is an expression of the form

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_1(a_i), \alpha_2(a_j), a_1, a_2, \ldots$$
$$\ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) .$$

There are two classes of type II decomposition. The first is when $\alpha_1(a_i) = a_i$ and $\alpha_2(a_j)$ is a one-place function which is not a constant or permutation function. A <u>permutation function</u> is one which simply relabels the input value. For example, the six three-valued permutation functions are given in the following table.

x	P_1	P_2	P_3	P_4	P_5	P_6
0	0	0	1	1	2	2
1	1	2	0	2	0	1
2	2	1	2	0	1	0

The second class of type II decomposition is when both α_1 and α_2 are nonconstant nonpermutation functions.

The class of type II decompositions with $\alpha_1(a_i) = a_i$ are expressions of the form

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_2(a_j), a_1, a_2, \ldots \\ \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) .$$

They are thus simple disjunctive decompositions where the set X simply consists of the values assumed by a_j in the definition of f. A variation of algorithm 3.1 could be used to find the compatible and incompatible pairs of X. An incompatibility graph $\hat{G}$ could be constructed and coloured and α_2 determined in a manner similar to that used in the proof of theorem 3.2. This process can be simplified by constructing $\hat{G}$ from G as follows.

Let $N(\hat{G})$ have one node for each value assumed by a a_j in the definition of f and label each node uniquely from these values. Let $L(\hat{G})$ be such that $\hat{n}_1$, $\hat{n}_2 \in N(\hat{G})$ are adjacent if, and only if, there is an $\ell \in L(G)$, $\ell = n_3n_4$, such that n_3 and n_4 are labeled x_s and x_t where x_s and x_t agree in the coordinate corresponding to a_i. $\hat{n}_1$ is labeled by the coordinate of n_3 corresponding to a_j and $\hat{n}_2$ is labeled by the coordinate of n_4 corresponding to a_j.

For example suppose G is given by figure 3.3.

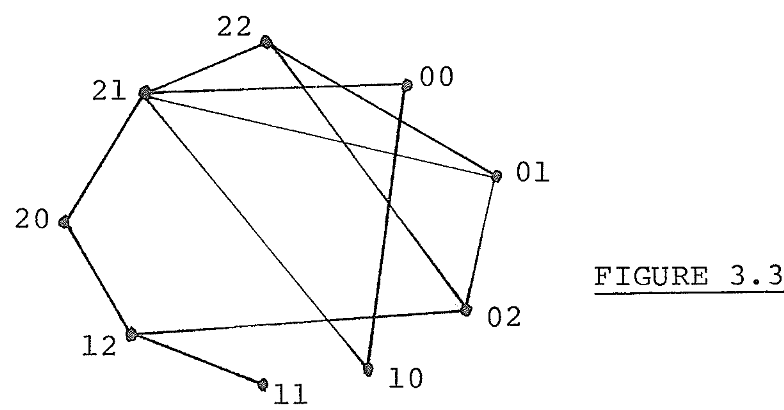


FIGURE 3.3

a_i corresponds to the first coordinate of these labels. Thus 00 and 01 agree in the coordinate corresponding to a_i. 00 and 10 do not. $N(\hat{G})$ consists of three nodes labeled 0, 1 and 2. Constructing $L(\hat{G})$ requires each pair of these nodes be examined in turn.

The nodes labeled 0 and 1 are adjacent in G since the nodes labeled 20 and 21 in G are adjacent. The nodes labeled 1 and 2 in $\hat{G}$ are adjacent since the nodes labeled 01 and 02 in G are adjacent. The nodes labeled 0 and 2 in $\hat{G}$ are not adjacent since 00 is not adjacent to 02 in G, 10 is not adjacent to 12 in G and 20 is not adjacent to 22 in G. $\hat{G}$ is thus given by figure 3.4.

FIGURE 3.4

Let $\hat{X}$ be the values assumed by a_j in the definition of f and let $\hat{Y}$ be the assignments assumed by $\{a_1, a_2, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n\}$ in the definition of f. Recall that X is the set of assignments assumed by $\{a_i, a_j\}$ in the definition of f, while Y is the set of assignments assumed by $\{a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n\}$ in the definition of f. The following theorem shows $\hat{G}$ as constructed above is the incompatibility graph for $\hat{X}$.

<u>Theorem 3.4.</u> Given $\hat{G}$ as described above, $\hat{x}_s \not\sim \hat{x}_t$, $\hat{x}_s$, $\hat{x}_t \in \hat{X}$, if, and only if, the nodes labeled $\hat{x}_s$ and $\hat{x}_t$ are adjacent in $\hat{G}$.

<u>Proof.</u> Suppose the nodes labeled $\hat{x}_s$ and $\hat{x}_t$ are adjacent in $\hat{G}$. By construction, there are nodes labeled $x_k = (q, \hat{x}_s)$ and $x_\ell = (q, \hat{x}_t)$ which are adjacent in G, hence $x_k \not\sim x_\ell$ and there is some $y \in Y$ such that $f[x_k, y] \neq f[x_\ell, y]$. It follows that $f[(q, x_s), y] \neq f[(q, x_t), y] \Rightarrow x_s \not\sim x_t$.

Suppose $\hat{x}_s \not\sim \hat{x}_t$. There is thus some $\hat{y} \in \hat{Y}$ such that $f[\hat{x}_s, \hat{y}] \neq f[\hat{x}_t, \hat{y}]$. Let q denote the value in $\hat{y}$ corresponding to a_i. $\hat{y}$ may be written as (q, y), $y \in Y$ and thus $f[\hat{x}_s,(q, y)] \neq f[\hat{x}_t,(q, y)]$. Rewriting we have $f[(q, \hat{x}_s), y] \neq f[(q, \hat{x}_t), y]$. It follows that the nodes labeled $(q, \hat{x}_2)$ and $(q, \hat{x}_t)$ are adjacent in G and by construction the nodes labeled $\hat{x}_s$, $\hat{x}_t$ are adjacent in $\hat{G}$. <u>QED</u>.

Let $\hat{p}$ be the number of nodes in $\hat{G}$.

<u>Theorem 3.5.</u> There exists a nontrivial type II decomposition of the form

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_2(a_j), a_1, a_2, \ldots \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) \quad (5)$$

if, and only if, $\chi(\hat{G}) < p$.

<u>Proof.</u> Suppose $\chi(\hat{G}) < \hat{p}$. Consider a $\chi(\hat{G})$ colouring of $\hat{G}$ and let c_t, $0 \leq t < \chi(\hat{G})$ denote the colours. For each $\hat{x}_t \in \hat{X}$ define $\alpha_2[\hat{x}_t] = s$ where for the colouring being considered the node labeled $\hat{x}_t$ is coloured c_s.

Since no two adjacent nodes have the same colour and since two nodes are adjacent if, and only if, there labels are incompatible, $\alpha_2[\hat{x}_s] = \alpha_2[\hat{x}_t] \Rightarrow \hat{x}_s \sim \hat{x}_t$ and by theorem 2.5 the decomposition exists. The decomposition is nontrivial since the range of α_2 is smaller than its domain as $\chi(\hat{G}) < p$.

Suppose a decomposition of the form (5) exists and suppose α_2 assumes k values. Clearly $k < p$. Choose k colours c_t, $0 \leq t < k$. For each $x_t \in X$ colour the corresponding node of G c_s where $\alpha_2[\hat{x}_t] = s$. Since $\alpha_2[\hat{x}_s] = \alpha_2[\hat{x}_t] \Rightarrow \hat{x}_s \sim \hat{x}_t$, and since nodes labeled with compatible elements of $\hat{X}$ are not adjacent, the above process defines a k-colouring of $\hat{G}$ and since $k < \hat{p}$, $\chi(\hat{G}) < \hat{p}$. QED.

The class of type II decompositions where both α_1 and α_2 are nonconstant, nonpermutation functions are expressions of the form

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_1(a_i), \alpha_2(a_j), a_1, a_2, \ldots$$
$$\ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) .$$

This is a special case of the multiple disjunctive decompositions considered in chapter 2. Recall that when f is total multiple disjunctive decompositions can be constructed by examining the simple disjunctive decompositions but when f is partial, this construction breaks down. The following example shows this is true even for the simple case where α_1 and α_2 are one-place functions.

Consider the function $f(a_1, a_2)$ given by

f	a_2 = 0	1	2
a_1 = 0	0	-	1
1	-	1	1
2	1	1	0

There exist decompositons

$$f(a_1, a_2) = g_1(\alpha_1(a_1), a_2)$$

and $f(a_1, a_2) = g_2(\alpha_2(a_2), a_1)$

where α_1 and α_2 are given by

a_1	α_1
0	0
1	0
2	1

a_2	α_2
0	0
1	0
2	1

There is, however, no g_3 such that

$$f(a_1, a_2) = g_3(\alpha_1(a_1), \alpha_2(a_2))$$

since $f(0,0) \neq f(1,1)$ but $\alpha_1(0) = \alpha_1(1)$ and $\alpha_2(0) = \alpha_2(1)$.

Multiple-disjunctive type II decompositions can not be determined directly from f but they can often be identified as a sequence of two simple type II decompositions.

8. TYPE III DECOMPOSITIONS

Type III decompositions are expressions of the form

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_1(a_i), \alpha_2(a_i, a_j), a_1, a_2, \ldots$$
$$\ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) .$$

There are two classes of type III decomposition

according as α_1 is a permutation function or not. When α_1 is a permutation function, it is always possible to choose $\alpha_1(a_i) = a_i$ since any other choice simply relabels the input assignments to g. This results in a decomposition of the form

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_2(a_i, a_j), a_1, a_2, \ldots \\ \ldots, a_{j-1}, a_{j+1}, \ldots, a_n)$$

which is a simple nondisjunctive decomposition.

<u>Theorem 3.6.</u> A type III decomposition with $\alpha_1(a_i) = a_i$ is nontrivial if, and only if, there exists some $x_s \sim x_t$, $x_s \in X$, such that x_s and x_t agree in the coordinate corresponding to a_i.

<u>Proof.</u> Suppose a nontrivial type III decomposition exists with $\alpha_1(a_i) = a_i$, α is defined by the 2-tuple $(a_i, \alpha_2(a_i, a_j))$. Since the decomposition is nontrivial α has a smaller range than its domain and there is some x_k, $x_\ell \in X$ such that $\alpha[x_k] = \alpha[x_\ell]$. Clearly, x_k and x_ℓ agree in the coordinate corresponding to a_i and $x_k \sim x_\ell$.

Suppose $x_s \sim x_t$ where x_s and x_t agree in the coordinate corresponding to a_i. For all $x_k \in X$, $k \neq t$, assign $\alpha_2[x_k]$ the value of the coordinate of x_k corresponding to a_j. Assign $\alpha_2[x_t]$ the value of the coordinate of x_s corresponding to a_j. Now $\alpha = (a_i, \alpha_2(a_i, a_j))$. By construction $\alpha[x_p] = \alpha[x_q] \Rightarrow x_p \sim x_q$ and hence by theorem 2.5 the decomposition exists. Since $\alpha[x_s] = \alpha[x_t]$ the range of α is smaller than its domain and

the decomposition is nontrivial. QED.

The above result is a condition for a type III decomposition with $\alpha_1(a_i) = a_i$ to be nontrivial. We now describe a process for determining α_2.

Construct a graph $\tilde{G}$ as follows. $N(\tilde{G}) = N(G)$. $L(\tilde{G}) = L(G) \cap \tilde{L}$ where $\tilde{L}$ consists of all unordered pairs of nodes $\tilde{n}_1, \tilde{n}_2 \in N(\tilde{G})$ such that $\tilde{n}_1$ and $\tilde{n}_2$ agree in the coordinate corresponding to a_i. For example, if G is given by figure 3.5 and a_i corresponds to the first coordinate of the labels of G, $\tilde{G}$ is given by figure 3.6.

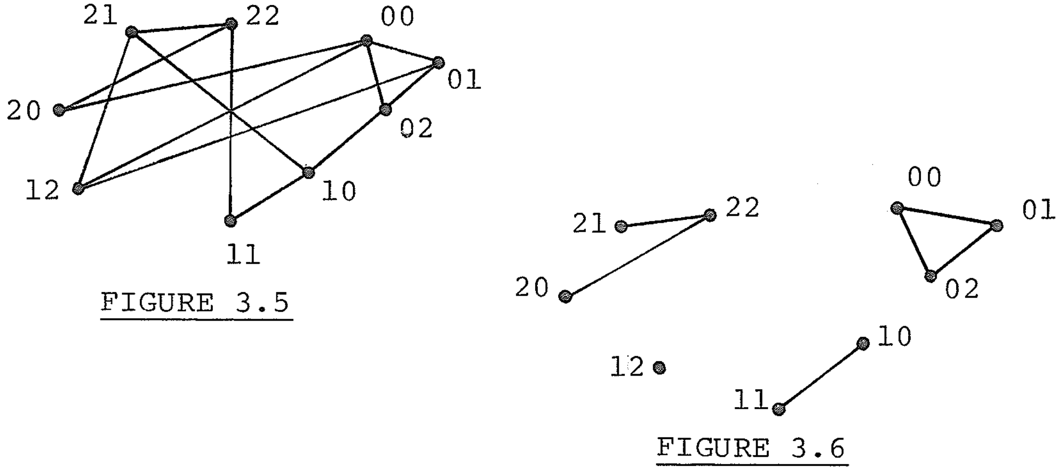


FIGURE 3.5

FIGURE 3.6

Consider a $\chi(\tilde{G})$ colouring of $\tilde{G}$ and let c_t, $0 \le t \le \chi(\tilde{G})$, denote the colours. For each $x_t \in X$, define $\alpha_2[x_t] = s$ where in the colouring being considered x_t is coloured c_s.

Theorem 3.7. Given α_2 constructed as above there exists a decomposition

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_2(a_i, a_j), a_1, a_2, \ldots \\ \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) .$$

Proof. α is defined by the 2-tuple $(a_i, \alpha_2(a_i, a_j))$. Consider $x_s \not\sim x_t$, $x_s, x_t \in X$. If x_s and x_t disagree in the coordinate corresponding to a_i, $\alpha[x_s] \neq \alpha[x_t]$. If x_s and x_t agree in the coordinate corresponding to a_i, the nodes labeled x_s and x_t are adjacent in $\tilde{G}$. This follows since $x_s \not\sim x_t$ implies the nodes labeled x_s and x_t are adjacent in G and since x_s and x_t agree in the coordinate corresponding to a_i, the adjacency is carried through to $\tilde{G}$. Since the nodes labeled x_s and x_t are adjacent in $\tilde{G}$, they must have different colours, hence $\alpha_2[x_s] \neq \alpha_2[x_t]$. Since $x_s \not\sim x_t \Rightarrow \alpha[x_s] \neq \alpha[x_t]$ by theorem 2.5 the decomposition exists. QED.

The second class of type III decompositions is when $\alpha_1(a_i)$ is a nonconstant, nonpermutation function. In considering the identification of decompositions from this class the question is whether the existence of the decompositions

$$f(a_1, a_2, \ldots, a_n) = g_1(\alpha_1(a_i), a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$$

and $$f(a_1, a_2, \ldots, a_n) = g_2(\alpha_2(a_i, a_j), a_1, a_2, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n)$$

implies the existence of a decomposition

$$f(a_1, a_2, \ldots, a_n) = g_3(\alpha_1(a_i), \alpha_2(a_i, a_j), a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) .$$

The following example disproves this notion.

Consider the function $f(a_1, a_2, a_3)$ given by

		a_1a_1								
		00	01	02	10	11	12	20	21	22
	0	0	0	0	0	0	0	1	2	1
a_3	1	2	1	2	1	1	1	2	0	2
	2	2	1	2	1	1	1	1	0	1

Examining this partition matrix results in the following incompatibility graph G.

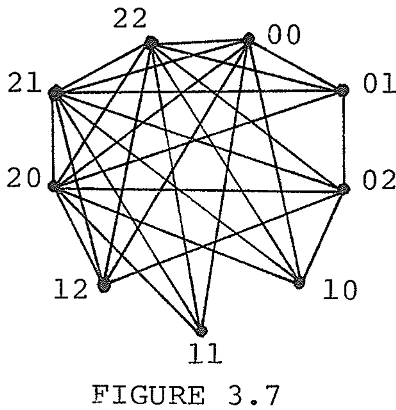


FIGURE 3.7

The nodes labeled 00, 01, 20 and 21 are mutually adjacent and $\chi(G) \geq 4$. There is thus no nontrivial type I decomposition of f for the partition $a_3 \mid a_1a_2$.

Consider a type II decomposition with $\alpha_1(a_1) = a_1$. Constructing $\hat{G}$ as described above we find $\hat{G}$ is given by figure 3.8.

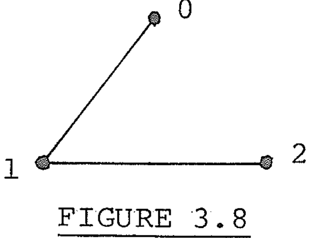


FIGURE 3.8

To see this note that the nodes labeled 00 and 01 are adjacent in G and the nodes labeled 01 and 02 are adjacent in G. However each pair labeled 00, 02; 10, 12; 20, 22 are not adjacent. It follows there is a decomposition

$$f(a_1, a_2, a_3) = g_1(\alpha_2(a_2), a_1, a_3)$$

where α_2 is given by the table

a_2	$\alpha_2(a_2)$
0	0
1	1
2	0

When $\alpha_1(a_2) = a_2$. $\hat{G}$ is given by figure 3.9,

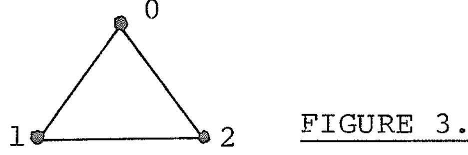


FIGURE 3.9

and there is no nontrivial decomposition of the form

$$f(a_1, a_2, a_3) = g(\alpha_2(a_1), a_2, a_3).$$

Now 01 ~ 11 hence there is a nontrivial type III decomposition with $\alpha_1(a_2) = a_2$. Constructing $\tilde{G}$ described above we find $\tilde{G}$ is given by figure 3.10.

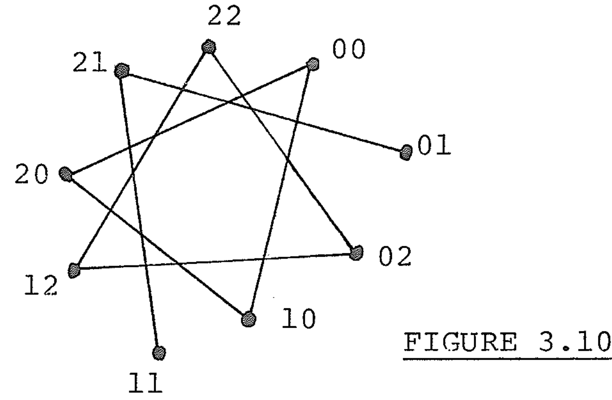


FIGURE 3.10

$\chi(\tilde{G}) \geq 3$ since the nodes labeled 00, 10 and 20 are mutually adjacent. One possible three-colouring is given by the sets {00,01,11,22}, {02,10}, {12,20,21} where two nodes have the same colour if, and only if, their labels are in the same set. This colouring results in a decomposition of the form

$$f(a_1, a_2, a_3) = g_2(\alpha_2(a_1, a_2), a_3)$$

where one possible choice for α_2 is given below.

	α_2	a_2 = 0	1	2
a_1	0	0	0	1
	1	1	0	2
	2	2	2	0

We have two decompositions

$$f(a_1, a_2, a_3) = g_1(\alpha_1(a_2), a_1, a_3)$$

and $f(a_1, a_2, a_3) = g_2(\alpha_2(a_1, a_2), a_3)$

However, there is no g_3 such that

$$f(a_1, a_2, a_3) = g_3(\alpha_2(a_1, a_2), \alpha_1(a_2), a_3) \ .$$

To see this, note that $f(0,0,0) \neq f(2,2,0)$. However, $\alpha_1(0) = \alpha_1(2)$ and $\alpha_2(0,0) = \alpha_2(2,2)$ and there is no way to assign g_3 consistently.

9. TYPE IV DECOMPOSITIONS

Type IV decompositions are the case where both α_1 and α_2 are true two-place functions. This is the most complex form of two-place decomposition. Thus for a given $f(a_1, a_2, \ldots, a_n)$ and pair of variables a_i

and a_j a type IV decomposition is assigned only when no simpler type of decomposition is possible.

Theorem 3.2 is a condition for the existence of a nontrivial two-place decomposition. If no type I, type II or type III decomposition can be assigned then a type IV decomposition is identified as follows.

Consider a many-valued function $f(a_1, a_2, \ldots, a_n)$ and pair of variables a_i and a_j. Suppose a nontrivial two-place decomposition exists which is not type I, II or III. Construct α in the same manner as α_2 was constructed in the proof of theorem 3.2. This α has the smallest possible range since it is determined from a minimal colouring of the incompatibility graph. Each element in the range of α is an integer k, $0 \le k < r^2$. For each $x_s \in X$, write $\alpha[x_s]$ as a two digit base r number and assign $\alpha_1[x_s]$ the value of the first digit and $\alpha_2[x_2]$ the value of the second digit.

For example, suppose f is three-valued, $X = \{00,01,10,11,12,20,22\}$ and α is such that

$$\alpha[00] = \alpha[11] = 0$$

$$\alpha[01] = \alpha[22] = 1$$

$$\alpha[10] = 2$$

$$\alpha[12] = \alpha[20] = 3$$

0 as a two digit base three number is 00. 1 is 01. 2 is 02. 3 is 10. Since $\alpha[00] = \alpha[11] = 0$, $\alpha_1[00] = \alpha_1[11] = 0$ and $\alpha_2[00] = \alpha_2[11] = 0$. Since $\alpha[01] = \alpha[22] = 1, \alpha_1[01] = \alpha_1[22] = 0$ and $\alpha_2[01] =$

$\alpha_2[22] = 1$. Similarly we find $\alpha_1[10] = 0$, $\alpha_2[10] = 0$, $\alpha_1[12] = \alpha_1[20] = 1$, $\alpha_2[12] = \alpha_2[20] = 0$.

10. A GRAPH COLOURING ALGORITHM

The above results establish tests for the existence of a number of forms of two-place decompositions of a many-valued function. Once it is known that a certain form of decomposition exists α_1, α_2, and g must be defined. Both the existence tests and the subsequent definition of the functions depend on the colouring of graphs. Several authors [6], [8], [34], [81] have considered this problem. The algorithms presented by these authors are costly in terms of both storage and execution time. The algorithm below, presented by the author in [37], is more economical in both these areas.

A path is an alternating sequence of nodes and lines beginning and ending with a node, in which each line is incident with the nodes immediately preceeding and following it, and in which all the nodes are distinct. A tree is a graph with exactly one path between each pair of nodes. One node will be distinquished as the root of the tree. The leaves of the tree are all nodes excluding the root which are incident with one line. Following normal mathematical convention, trees will be drawn with the root at the top and the leaves at the bottom, e.g. the tree below.

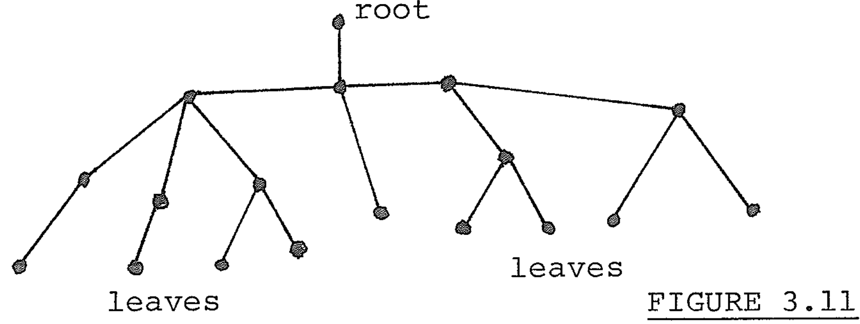


FIGURE 3.11

The unique colourings of a graph up to renaming of the colours can be represented by a labeled tree. A labeled tree is one whose nodes are distinquished by attaching a unique label to each node. A tree representing the colourings of a graph is termed a colouring tree.

Consider a graph G with p nodes $n_0, n_1, \ldots, n_{p-1}$. Let $c_0, c_1, \ldots, c_{p-1}$ denote p unique colours. Suppose the nodes $n_0, n_1, \ldots, n_{t-1}$, $t < p$, have been coloured using $c_0, c_1, \ldots, c_{\ell-1}$, $\ell \leq t$. The node n_t can be coloured c_k, $0 \leq k \leq \ell-1$, if, and only if, there is no n_m, $0 \leq m \leq t-1$, such that n_m and n_t are adjacent and n_m is coloured c_k. In addition, n_t can be coloured a new colour c_ℓ. Since we are only concerned with colourings up to renaming of the colours, this exhausts all choices of colours for n_t for the given colouring of $n_0, n_1, \ldots, n_{t-1}$. Beginning by assigning n_0 the colour c_0 and then applying the above result repeatedly, a colouring tree can be constructed. The process is best described by working an example.

Consider the graph G given by figure 3.12.

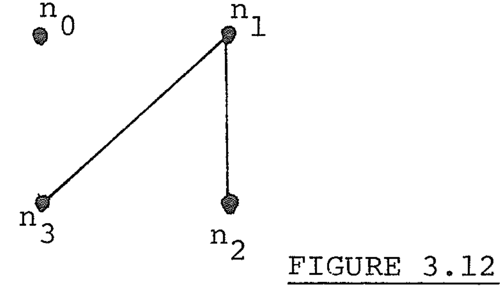


FIGURE 3.12

The root of the colouring tree is given in figure 3.13,

• (n_0, c_0) FIGURE 3.13

where the label (n_0, c_0) indicates the node n_0 in G is to be coloured c_0. For the vertex n_1 there are two choices. Either n_1 can be coloured c_0 since it is not adjacent to n_0 or it can be coloured a new colour c_1. These choices are represented by the tree below.

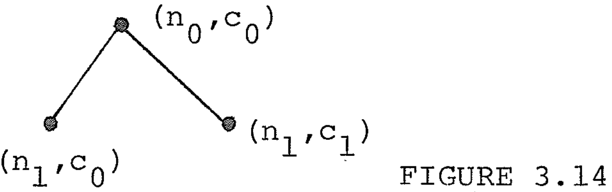


FIGURE 3.14

Each path from the root to a leaf in this tree represents a colouring of the nodes n_0, n_1 of G. Consider the left path. Since n_1 is coloured c_0, the only choice is to colour n_2 a new colour c_1. This gives the following tree.

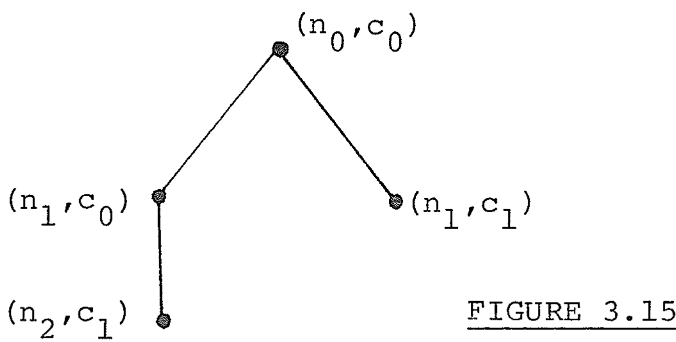


FIGURE 3.15

When n_0 is coloured c_0 and n_1 is coloured c_1, n_2 may be coloured c_0 but not c_1. n_2 can also be coloured a new colour c_2. Thus we have the tree:

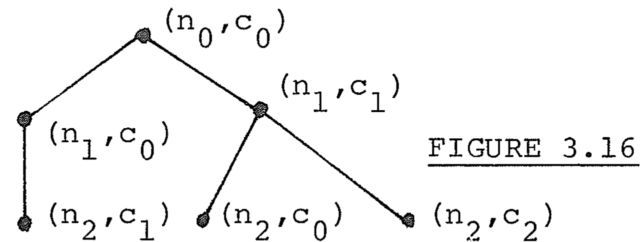


FIGURE 3.16

In this tree there are three paths from the root to a leaf representing three unique colouring of the nodes n_0, n_1, n_2 of G. Applying the above techniques to n_3 the complete colouring tree, figure 3.17, is found.

Each path from the root to a leaf in this tree represents a colouring of G. It is easily verified that every unique colouring of G up to renaming of the colours is represented. The darker paths represent minimal colourings and indicate $\chi(G) = 2$. Note that in constructing a colouring tree the lines down from a node lead to nodes assigning colours in ascending order from left to right.

In the above example, the leftmost path represents a minimal colouring. This is the case for many graphs but is not always true. Consider the graph G given by figure 3.18.

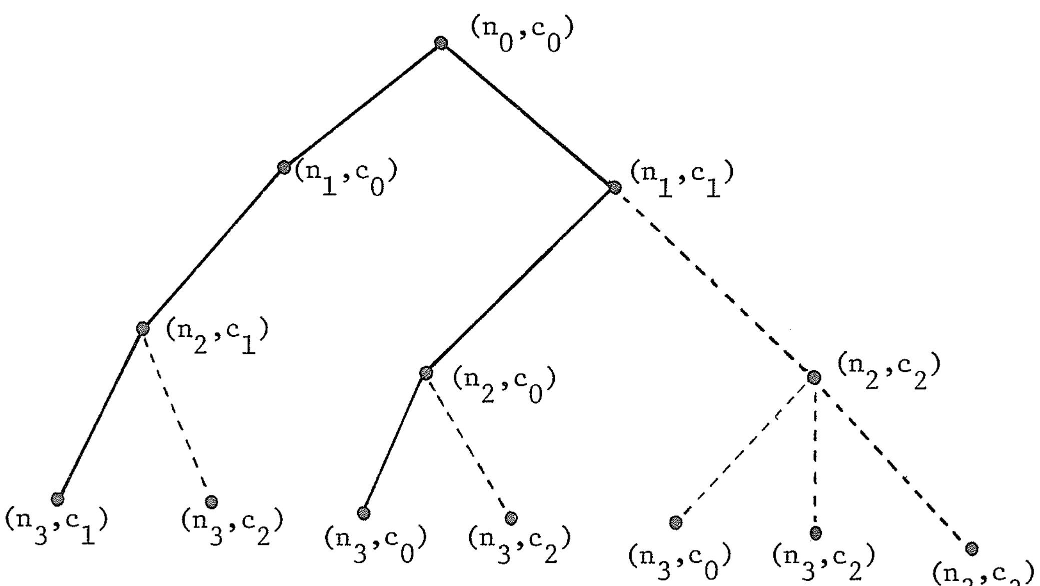


FIGURE 3.17

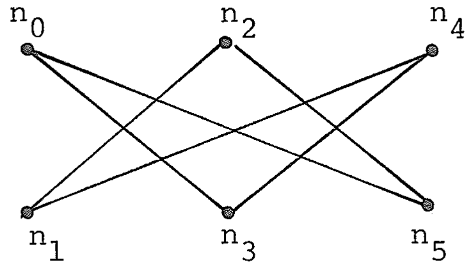


FIGURE 3.18

If the colouring tree is constructed, the leftmost path is found to be as shown in figure 3.19.

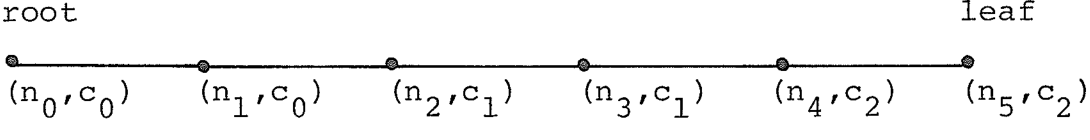


FIGURE 3.19

This colouring uses three colours yet clearly $\chi(G) = 2$.

The chromatic number of a graph G can be determined by examining the paths of the corresponding colouring tree. The following rules examine the paths in a tree from left to right:

i) start at the root and go down the leftmost line;

ii) when coming down a line to a node which is not a leaf, go down the leftmost line;

iii) when a leaf is reached, go back up the line leading to that leaf;

iv) when coming up a line to a node, go down the next line to the right or if there are no more lines to the right, go up the line to the next higher node;

v) stop after returning over the rightmost line

leading from the root.

It is an easy matter to count the number of colours assigned in each path. The minimum of these is $\chi(G)$ and the corresponding path defines a $\chi(G)$ - colouring.

Two simple observations greatly reduce this search. Suppose we have just reached a leaf on a path representing a colouring of G which uses k colours. We can immediately go back up this path past the highest node which assigns the colour c_{k-1}. The reason for this is that any path containing this node represents a colouring using k or more colours. Similarly, it is unnecessary to visit any further nodes which assign a colour c_ℓ, $\ell \geq k-1$, since a path containing such a node represents a colouring with k or more colours.

The graph G given by figure 3.20.

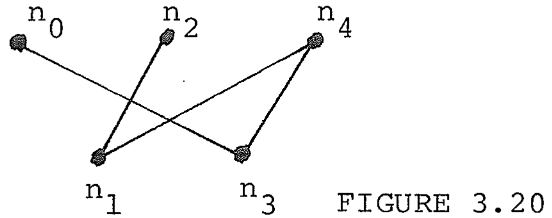


FIGURE 3.20

has the colouring tree given in figure 3.21. The arrows indicate the nodes visited in a reduced search. In the reduced search, the last path from the root to a leaf which is fully completed represents a minimal colouring.

It is unnecessary to construct the colouring tree. Instead, we need only keep track of the path from the root to the node currently being visited. This path is

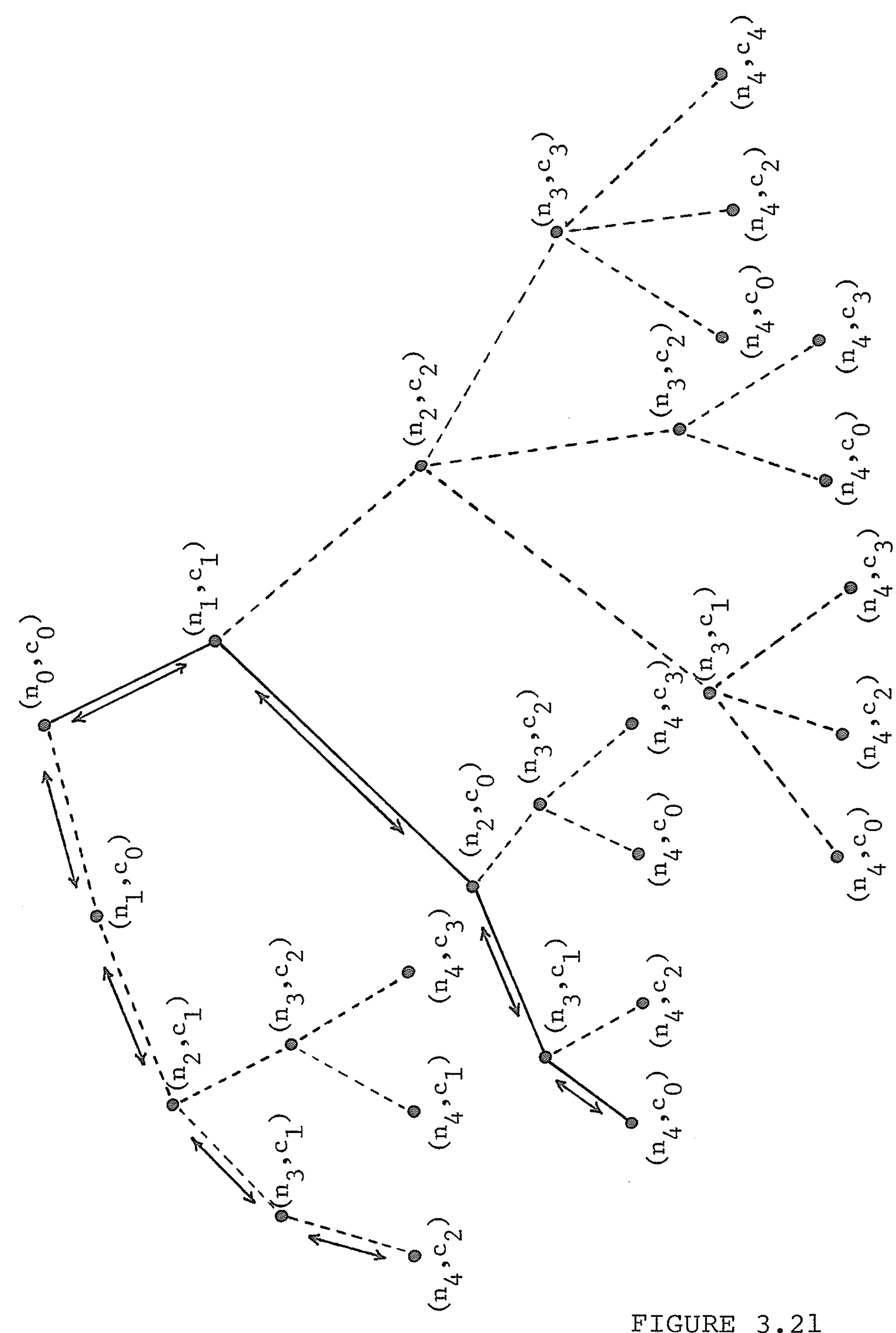


FIGURE 3.21

uniquely defined by the colours currently assigned to the nodes of G. The values $\gamma_0, \gamma_1, \ldots, \gamma_{p-1}$ indicate the current colouring of G. If $\gamma_i = -1$, m_i is uncoloured. If $0 \leq \gamma_i \leq p-1$, n_i is coloured c_{γ_i}. ψ_i is one less than the number of colours currently used to colour the vertices $n_0, n_1, \ldots, n_{i-1}$. These values are useful for retreating back up the path when a colouring is found. If the colouring uses k colours, we simply back up to the first ψ_i such that $\psi_i < k$. At each point in the algorithm t indicates the vertex of G currently being considered.

The sets $C_0, C_1, \ldots$ are termed <u>colour sets</u>. At each point is the execution of the algorithm, the set C_i contains all n_j such that n_j is coloured c_i. The graph to be coloured is represented by the sets $A_0, A_1, \ldots, A_{p-1}$ where A_i contains all n_j such that n_j and n_i are adjacent in G. The advantage of this representation is that we can colour n_i with c_j if, and only if, $A_i \cap C_j = \phi$. The representation of these sets and the execution of this test are tasks quite suitable to a computer.

<u>Algorithm 3.2.</u>

1. $\gamma_0 \leftarrow 0;\ \psi_0 \leftarrow 0;\ t \leftarrow 1;\ C_0 \leftarrow \{n_0\};\ min \leftarrow p$
2. $\gamma_i \leftarrow -1,\ C_i \leftarrow \phi,\ 1 \quad i < p$
3. find k the least integer such that $\gamma_t < k$ $\psi_{t-1} + 1$ and $C_k \cap A_t = \phi$

4. if $k \geq$ min go to 15
5. if $\gamma_t \neq -1$ remove n_t from C_t
6. $\gamma_t \leftarrow k$; add n_t to C_k
7. $\psi_t \leftarrow \text{Max}(\delta_{t-1}, k)$
8. $t \leftarrow t+1$
9. if $t \leq p-1$ go to 3
10. n_i coloured c_{γ_i}, $0 \leq i < p$ is a colouring of G
11. min $\leftarrow \psi_{p-1}$
12. $t \leftarrow t-1$
13. if $t = 0$ stop
14. if $\psi_t \leq$ min -1 and $\delta_t \leq$ min -2 and
    $\delta_t \leq \psi_{t-1}$ go to 3
15. remove n_t from C_{δ_t}
16. $\delta_t \leftarrow 0$
17. go to 12

For certain graphs this algorithm produces more than one colouring. The last of these colourings is minimal.

11. ASSIGNMENT OF α_1 AND α_2

For a given many-valued function and a_i, a_j pair, if a two-place decomposition exists it is not unique. There is often more than one type of decomposition and for each of these types there are a number of choices for α_1 and α_2. We would like to identify which of these decompositions results in the least complexity in the final switching circuit. This requires we identify the decomposition involving the fewest

arguments to g and the least amount of hardware. The first criterion is straightforward. The second is much more involved and is closely related to the way the functions are to be realized in hardware.

Several many-valued hardware schemes have appeared in the literature [15], [43], [80]. These schemes employ various technologies to implement some set of primitive operators. More complex functions are realized using some number of these operators. The two-place decomposition techniques can be used to express a many-valued function as a composition of one and two-place functions. We are thus concerned with how primitive hardware devices can be used to realize two place functions.

There are $(r^{r^2} - r)$ true one or two-place r-valued functions. It is impossible to keep a table of the realizations for each of these functions. This implies a realization must be determined each time a function is used. If a minimal realization is sought this would require a great deal of computing and would result in a very hardware-dependant synthesis technique. An alternative approach is to realize each function in some predetermined form.

For example, the condition disjunction due to Church [7] can be generalized to r values to give a function denoted

$$[a, b_0, b_1, \ldots, b_{r-1}]$$

whose value is b_i when $a = i$, $0 \leq i \leq r-1$. This function can realize any one-place r-valued function by applying the appropriate constants to the b_i. A two-place r-valued function can be realized as

$$f(a_1, a_2) = [a_1, f_0(a_2), f_1(a_2), \ldots, f_{r-1}(a_2)] \quad (6)$$

where $f_i(a_2)$ is the value of $f(a_1, a_2)$ when $a_1 = i$. This reduces the problem of realizing $f(a_1, a_2)$ to the problem of realizing the generalized condition disjunction and the $f_i(a_2)$. The realization of the generalized condition disjunction is the same for all r-valued functions. The $f_i(a_2)$ are one-place functions. Certainly, for $r = 3$ or $r = 4$ and possibly for $r = 5$, it would be reasonable to keep a table of realizations of the one-place r-valued functions. A realization of a two-place function could thus be determined fairly easily.

Sobociński [69] has taken the idea of a pre-defined form further. A universal decision element is one which can realize any one or two-place function by an appropriate assignment to the arguments of the element. Sobociński considered the two-valued case and restricted the allowed arguments to the constants 0, 1 and the variables of the function to be realized.

Loader [31], [32], Rose [57] and Muzio and Miller [46], [47] have considered various many-valued universal decision elements particularly the three-valued case. For a number of these elements the

allowed arguments include certain one-place functions in addition to the constants and variables. For example, Muzio and Miller [46] present a three-valued universal decision element with seven arguments which are allowed to be constants, variables or the Post negations [52] of one of the variables.

The universal decision element has two major advantages. First, each one or two-place function is realized by the same hardware configuration except possibly for operators on the inputs. This makes the switching circuit easier to construct since the designer can treat the universal elements as 'black boxes' considering only their external connections. This is much easier than constructing a unique realization for each function.

The second advantage is that constructing the switching circuit using universal decision elements greatly reduces the complexity of a synthesis algorithm based on decomposition. For a two-place decomposition there are a number of choices for α_1 and α_2. If the switching circuit is realized using a hardware scheme where a unique realization is to be found for each function, the cost and the number of connections may vary widely for the choices of α_1 and α_2. This would imply the choices should be examined to determine a minimal or a nearly minimal choice. If a universal element is used the cost and number

of connections is the same for all choices except possibly for operators on certain inputs. It is quite reasonable to choose the functions arbitrarily. This means that for each incompatibility graph only one minimal colouring and a single assignment of α_1 and α_2 derived from that colouring need be considered.

In a sequence of decompositions representing a switching circuit the arguments of any one-place nonpermutation functions must be arguments of the initial function to be realized since arguments introduced by decompositions i.e. the α_i, all have minimal ranges. The one-place functions are all arguments to two-place functions. If universal elements are used any one-place functions are superfluous since they are easily incorporated into the following two-place functions. While this alters the two-place function the cost of realizing it does not change. The overall circuit is in fact simpler since the one-place function need not be realized separately. When universal elements are to be used decompositions involving one-place functions other than the variable itself are ignored.

When one-place functions are ignored there are three forms of two-place decomposition

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_2(a_i, a_j), a_1, a_2, \ldots \\ \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n)$$

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_2(a_i, a_j), a_1, a_2, \ldots \\ \ldots, a_{j-1}, a_{j+1}, \ldots, a_n)$$

$$f(a_1, a_2, \ldots, a_n) = g(\alpha_1(a_i, a_j), \alpha_2(a_i, a_j), a_1, a_2, \ldots$$
$$\ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n).$$

These are type I, type III and type IV decompositions respectively. They are listed in order of perference. The first employs a single true two-place function and n-1 arguments to g. The second employs a single true two-place function and n arguments to g. The third employs two true two-place functions and n arguments to g. Conditions for the existence of each of these decompositions are given by theorems 3.2, 3.6 and 3.1 respectively.

11. Hardware Realizations

Switching circuits are constructed using basic hardware elements which perform relatively simple operations. The theoretical results above assume an appropriate universal decision element is available. There is thus a gap between our theoretical results and the practical design problem.

One approach to solving this problem is to extend our results to handle basic hardware elements directly. This would require a different technique for assigning α_1 and α_2 in a decomposition. Such a technique would have to examine the possibilities for α_1 and α_2 and choose those with the simplest realization in terms of the basic hardware available.

An alternative approach is to determine a realization of the universal decision element of interest in terms of

the basic hardware available. This is the easier of the two solutions, since our theoretical results need not be modified. It's practicality is questionable. It realizes each two-place many-valued function in the same way and will thus often waste hardware. However, with the advent of many-valued integrated circuits [43], this regularity of realization may in fact prove to be an advantage.

It is beyond the scope of this thesis to consider the hardware realization problem in detail. The author is currently examining the approaches mentioned above. The results of this research will clearly have an effect on the usefulness of the decomposition results as well as the direction of future theoretical work in this area.

CHAPTER 4

A Fast Method for Determining the Two-Place Decompositions of a Two-Valued Function

1. INTRODUCTION

A number of authors have considered the decomposition of two-valued functions. Ashenhurst [2] and Curtis [9] have developed a complete theory of decomposition of total two-valued functions. These results were considered in chapter 2. The techniques are extremely time consuming and cannot be extended to partial functions.

Roth and Karp [59] and Karp *et al* [60] have presented an approach which can be used for partial functions. The decompositions considered take the form

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

where $A_\lambda \cup A_\mu = A$, $A_\lambda \cap A_\mu = \phi$ and each α_i is a vertex function i.e. a function which is 1 or 0 for exactly one input condition. This restriction is justified since commonly available two-valued devices implement one of the vertex functions AND, OR, NAND or NOR.

Karp *et al* [60] have implemented a vertex decomposition technique on a computer. Unfortunately, their description is quite vague. It is impossible to tell exactly what form of decompositions are allowed

but it is interesting to note that in the examples given, all but one gate have two inputs.

Two-place decompositions of partial two-valued functions have been considered by Barnard and Holman [3] and by the author [36]. These techniques are much more efficient than those due to Roth and Karp [59] and Karp et al [60]. Barnard and Holman present heuristic decomposition tables which identify simple disjunctive or nondisjunctive two-place decompositions where α is a vertex function. The outstanding feature of this work is that almost all the computation is carried out in parallel for all pairs of variables. Only very simple tests must be carried out individually for each pair.

The method presented by the author in [36] allows for the equivalence and exclusive - OR and also for decompositions where both α_1 and α_2 are true two-place functions. For each pair of variables the compatible and incompatible elements of X are found as in chapter 3. Since for a two-valued function there are at most 4 elements in X, there are 64 possible incompatibility graphs. This figure is found by assuming input configurations not appearing in X are included in the graph as compatible with all others. For each possible graph, the 'best' decomposition is kept in a table. In determining the two-place decompositions of a

function this table is indexed by each graph encountered. This method has the disadvantage that the incompatibility graph for each pair of variables must be determined separately.

The technique developed below combines the advantages of these two approaches. Most of the required computation is performed in parallel and the two-place decompositions not considered by Barnard and Holman [3] are permitted. The method is much more efficient than those due to Roth and Karp [59] and Karp _et al_ [60] and is in practice just as useful since there is no evidence the latter techniques can be applied for other than two-place decompositions.

For the rest of this chapter, a function is two-valued unless otherwise indicated. Functions will be represented in a different form from the matrix representation introduced in chapter 3. The representation chosen is due to Roth [58]. It is more compact and leads to a clearer theoretical development.

2. NOTATION AND DEFINITIONS

A _cube_ is an n-tuple of 0's, 1's, x's and ϕ's. If the cube contains no x's or ϕ's it is termed a _vertex_. A vertex simply represents an assignment of values to the variables a_1, a_2, ..., a_n. A cube containing no ϕ's is a representation of a set of vertices. The elements of this set are found by replacing the x's

in the cube by 0's and 1's in all possible ways.

$f(a_1, a_2, \ldots, a_n)$ is a vertex function if, and only if, it assumes the value 0 or 1 for exactly one input condition. This input condition is termed the distinguished vertex.

We shall use E, C, X, T^o, T^1, U and V to denote sets of cubes and shall denote their elements as e_k, c_k, x_k, t_k^o, t_k^1, u_k and v_k, respectively. A second subscript will be used to denote a specific coordinate of a cube eg. c_{kp} is the pth coordinate of the kth cube of C. The set of vertices represented by the cube c_k is denoted $[c_k]$. $f[c_k]$ is defined if, and only if, f assumes the same value for every vertex in $[c_k]$.

We define three operators over the set $\{\phi, 0, 1, x\}$:

$\cup$	ϕ	0	1	x
ϕ	ϕ	0	1	x
0	0	0	x	x
1	1	x	1	x
x	x	x	x	x

$\wedge$	0	1	x
0	0	ϕ	0
1	ϕ	1	1
x	0	1	x

ζ	0	1	x
0	ϕ	0	0
1	1	ϕ	1
x	1	0	x

These operations are extended to cubes as follows

$$c_p \cup c_q = \{c_{p1} \cup c_{q1}, c_{p2} \cup c_{q2}, \ldots\}$$
$$c_p \wedge c_q = \{c_{p1} \wedge c_{q1}, c_{p2} \wedge c_{q2}, \ldots\}$$
$$c_p \zeta c_q = \{c_{p1} \zeta c_{q1}, c_{p2} \zeta c_{q2}, \ldots\}$$

In addition we define

$c_p \cap c_q = \phi$ if $c_{p\ell} \wedge c_{q\ell} = \phi$ for any ℓ,

$c_p \cap c_q = c_p \wedge c_q$ otherwise.

The distance between c_p and c_q, denoted $\Delta(c_p,c_q)$, is the number of distinct ℓ such that $c_{p\ell} \wedge c_{q\ell} = \phi$. We write $c_{p\ell} \subseteq c_{q\ell}$ if, and only if, $c_{p\ell} \wedge c_{q\ell} = c_{p\ell}$ and $c_p \subseteq c_q$ if, and only if, $c_p \cap c_q = c_p$.

Given i and j we define $\lambda(c_p) = \{c_{pi},c_{pj}\}$ and $\mu(c_p) = \{c_{p1},c_{p2},\ldots,c_{pi-1},c_{pi+1},\ldots,c_{j-1},c_{j+1},\ldots,c_{pn}\}$. $\lambda(c_p)$ and $\mu(c_p)$ are themselves cubes and we denote their coordinates as $\lambda(c_p)_k$ and $\mu(c_p)_k$. $[\lambda(c_p)]$ and $[\mu(c_p)]$ are the sets of vertices represented by $\lambda(c_p)$ and $\mu(c_p)$.

A function $f(a_1,a_2,\ldots,a_n)$ is degenerate if there exists a function g such that

$$f(a_1,a_2,\ldots,a_n) = g(b_1,b_2,\ldots,b_m)$$

where $m < n$, and $b_i \in \{a_1,a_2,\ldots,a_n\}$, $1 \le i \le m$. The degree of f, denoted $\delta(f)$, is the minimum m so that such a g exists.

Consider an arbitrary function $f(a_1,a_2,\ldots,a_n)$. A two-place decomposition is an expression of the form

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_1(a_i,a_j),\alpha_2(a_i,a_j),a_1,a_2,\ldots \\ \ldots,a_{i-1},a_{i+1},\ldots,a_{j-1},a_{j+1},\ldots,a_n) \qquad (1)$$

where α_1 and α_2 are two-valued functions. The decomposition is nontrivial if, and only if, g is defined for fewer input conditions than f. We insist that g be a nondegenerate function, hence, $\alpha_1(a_i,a_j) \neq \alpha_2(a_i,a_j)$ and at least one of α_1 or α_2 is a true two-place function. Also α_1 and α_2 may be interchanged as this simply results

in a re-labeling of the inputs to g.

There are three types of nontrivial two-place decompositions of a nondegenerate two-valued function.

Simple Disjunctive

$$f(a_1,a_2,\ldots a_n) = g(\alpha_2(a_i,a_j),a_1,a_2,\ldots \ldots,a_{i-1},a_{i+1},\ldots,a_{j-1},a_{j+1},\ldots,a_n) \quad (2)$$

Simple Nondisjunctive

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_2(a_i,a_j),a_1,a_2,\ldots \ldots,a_{i-1},a_{i+1},\ldots,a_n) \quad (3)$$

Complex Disjunctive

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_1(a_i,a_j),\alpha_2(a_i,a_j),a_1,a_2,\ldots \ldots,a_{i-1},a_{i+1},\ldots,a_{j-1},a_{j+1},\ldots,a_n) \quad (4)$$

In expressions (2), (3) and (4) α_1 and α_2 denote true two-place functions.

In the simple disjunctive case $\alpha_1(a_i,a_j)$ is a constant and is removed from g. In the simple non-disjunctive case $\alpha_1(a_i,a_j) = a_j$. Since a_i and a_j may be interchanged and since a_j in (3) may be replaced by $\bar{a}_j$, these three types exhaust all possibilities.

The complex disjunctive decompositions considered here are not the same as those discussed in chapter 2 since α_1 and α_2 have the same arguments. We adopt this term since the decomposition has two true two-place functions and since the arguments to g and the α_i are disjoint.

3. DECOMPOSITION TESTS

Let E denote the set of all vertices for which $f(a_1, a_2, \ldots, a_n)$ is defined. Given i and j let X denote the set of all unique $\lambda(e_\ell)$. We restate the definition of compatibility in the notation of this chapter.

Definition. x_p and x_q are compatible (denoted $x_p \sim x_q$) if, and only if, for all e_ℓ, e_m such that $\lambda(e_\ell) = x_p$, $\lambda(e_m) = x_q$ and $\mu(e_\ell) = \mu(e_m)$, $f[e_\ell] = f[e_m]$; otherwise x_p and x_q are incompatible (denoted $x_p \not\sim x_q$).

Given $\alpha_1(a_i, a_j)$ and $\alpha_2(a_i, a_j)$ we shall say that g exists if, and only if, expression (1) holds for every vertex for which f is defined. The following criterion is a special case of theorem 2.1.

Criterion. Given $\alpha_1(a_i, a_j)$ and $\alpha_2(a_i, a_j)$, g exists if, and only if, for all x_p, $x_q \in X$,

$$\alpha_1[x_p] = \alpha_1[x_q] \text{ and } \alpha_2[x_p] = \alpha_2[x_q] => x_p \sim x_q,$$

or equivalently

$$x_p \not\sim x_q => \alpha_1[x_p] \neq \alpha_1[x_q] \text{ or } \alpha_1[x_p] \neq \alpha_2[x_q].$$

Clearly, we could examine all the incompatible pairs of elements of X and determine if g exists. This is the approach taken by Roth and Karp [59] and by the author in a previous study [36]. Simpler tests can be developed if each type of decomposition is considered in turn.

We assume $\alpha_1(a_i, a_j)$ and $\alpha_2(a_i, a_j)$ are given.

$\beta[x_k]$ will denote a vertex function with distinguished vertex x_k. There are two for each value of k.

Simple Disjunctive

Theorem 4.1. If $\delta(\alpha_1) = 0$ and $\alpha_2 = \beta[x_k]$, g exists if, and only if, for all $x_p \not\sim x_q$, p = k or q = k.

Proof. Suppose for all $x_p \not\sim x_q$, p = k or q = k. Consider any $\ell \neq m$ such that $\alpha_2[x_\ell] = \alpha_2[x_m]$. Clearly, $\ell \neq k$ and $m \neq k$ and therefore $x_\ell \sim x_m$ and g exists.

Suppose g exists and consider any $x_p \not\sim x_q$. Since $\delta(\alpha_1) = 0$, $\alpha_2[x_p] \neq \alpha_2[x_q]$. $\alpha_2 = \beta[x_k]$ and thus either p = k or q = k. QED.

Theorem 4.2. If $\delta(\alpha_1) = 0$ and α_2 is the equivalence or nonequivalence, g exists if, and only if, 00 ~ 11 and 01 ~ 10.

Proof. Suppose 00 ~ 11 and 01 ~ 10. Consider any $\ell \neq m$ such that $\alpha_2[x_\ell] = \alpha_2[x_m]$. Clearly if $x_\ell \in \{00,11\}$, $x_m \in \{00,11\}$ and if $x_\ell \in \{01,10\}$, $x_m \in \{01,10\}$. Whichever, $x_\ell \sim x_m$ and g exists.

Suppose g exists, $\alpha_2[11] = \alpha_2[00]$, $\alpha_2[01] = \alpha_2[10]$ and $\delta(\alpha_1) = 0$, hence, 00 ~ 11 and 01 ~ 10. QED.

Simple Nondisjunctive

Let $\gamma(1) = i$ and $\gamma(2) = j$.

Theorem 4.3. If $\alpha_2 = \beta[x_k]$ and $\alpha_1 = a_{\gamma(p)}$ or $\alpha_1 = \bar{a}_{\gamma(p)}$, $p \in \{1,2\}$, g exists if, and only if, for all $x_\ell \not\sim x_m$ such that $x_{\ell p} = x_{mp}$, $x_{\ell p} = x_{kp}$.

Proof. Suppose for all $x_\ell \not\sim x_m$ such that $x_{\ell p} = x_{mp}$, $x_{\ell p} = x_{kp}$. Consider any $x_r \not\sim x_s$. If $x_{rp} \neq x_{sp}$, $\alpha_1[x_r] \neq \alpha_1[x_s]$. If $x_{rp} = x_{sp}$, $\alpha_2[x_r] \neq \alpha_2[x_s]$ as $x_{rp} = x_{sp}$ and then either r = k or s = k. Consequently, g exists.

Suppose g exists. Consider $x_\ell \not\sim x_m$ such that $x_{\ell p} = x_{mp}$. It follows that $\alpha_2[x_\ell] \neq \alpha_2[x_m]$. Since $\alpha_2 = \beta[x_k]$, $x_{\ell p} = x_{kp}$. QED.

Complex Disjunctive

Theorem 4.4. If $\alpha_1 = \beta[x_p]$ and $\alpha_2 = \beta[x_q]$, g exists if, and only if, for some ℓ, m, $x_\ell \sim x_m$ where p, q, ℓ and m are mutually distinct.

Proof. Suppose $x_\ell \sim x_m$. Consider $r \neq s$ such that $\alpha_1[x_r] = \alpha_1[x_s]$ and $\alpha_2[x_r] = \alpha_2[x_s]$. Clearly, p, q, r and s are mutually distinct as $\alpha_1 = \beta[x_p]$ and $\alpha_2 = \beta[x_q]$. It follows that either $r = \ell$ and s = m or r = m and $s = \ell$. Whichever, $x_r \sim x_s$ and g exists.

Suppose g exists. Consider $r \neq s$ such that $\alpha_1[x_r] = \alpha_1[x_s]$ and $\alpha_2[x_r] = \alpha_2[x_s]$. It follows that $x_r \sim x_s$ and r, s, p and q are mutually distinct. QED.

It can be easily verified that if $x_\ell \sim x_m$ and $x_{\ell p} = x_{mp}$ then either α_1 or α_2 is redundant in any possible complex disjunctive decomposition. 00 ~ 11 and 01 ~ 10 are the only relations which lead to a nondegenerate complex disjunctive decomposition.

Let C denote a set of cubes such that $e_\ell \in [c_m]$

for all $e_\ell \in E$ and $f[c_k]$ is defined for all $c_k \in C$. We write $\lambda(c_\ell) \not\sim \lambda(c_m)$ if, and only if, $e_p \not\sim e_q$ for all $e_p \in [\lambda(c_\ell)]$, $e_q \in [\lambda(c_m)]$. The following lemmata are obvious.

Lemma 4.1. Given k, $p = k$ or $q = k$ for all $x_p \not\sim x_q$ if, and only if, for all $\lambda(c_\ell) \not\sim \lambda(c_m)$, $[\lambda(c_\ell)] = x_k$ or $[\lambda(c_m)] = x_k$.

Lemma 4.2. Given k and p, $x_{\ell p} = x_{kp}$ for all $x_\ell \not\sim x_m$ such that $x_{\ell p} = x_{mp}$ if, and only if, for all $\lambda(c_r) \not\sim \lambda(c_s)$ such that $\lambda(c_r)_p \wedge \lambda(c_s)_p = \phi$, $\lambda(c_s)_p \wedge \lambda(c_s)_p = x_{kp}$.

The sets T^0, T^1, U and V are constructed from C as follows:

Algorithm 4.1

1. Initially set $t_\ell^0 = t_\ell^1 = u_\ell = v_\ell = \phi$ for all $1 \le \ell \le n$.

2. For all c_p, c_q such that $f[c_p] \ne f[c_q]$ let k be the smallest integer such that $c_{pk} \wedge c_{qk} = \phi$ and suppose with no loss of generality $c_{pk} = 1$.

(i) If $\Delta(c_p, c_q) = 1$

$$t_k^0 \leftarrow t_k^0 \cup c_q$$

$$t_k^1 \leftarrow t_k^1 \cup c_p$$

$$u_k \leftarrow u_k \cup (c_p \wedge c_q)$$

$$v_k \leftarrow v_k \cup (c_p \zeta c_q).$$

(ii) If $\Delta(c_p, c_q) = 2$

$$t^o{}_{kk} \leftarrow t^o{}_{kk} \cup c_{qk}$$

$$t^1{}_{kk} \leftarrow t^1{}_{kk} \cup c_{pk}$$

$$t^o{}_{k\ell} \leftarrow t^o{}_{k\ell} \cup c_{q\ell}$$

$$t^1{}_{k\ell} \leftarrow t^1{}_{k\ell} \cup c_{p\ell}$$

$$V_{k\ell} \leftarrow V_{k\ell} \cup (c_{p\ell} \zeta c_{q\ell}).$$

(iii) If $\Delta(c_p, c_q) \geq 3$ ignore the pair.

The sets T^o, T^1, U and V have some interesting properties.

Theorem 4.5. Given x_k, $[\lambda(c_p)] = x_k$ or $[\lambda(c_q)] = x_k$ for all $\lambda(c_p) \neq \lambda(c_q)$ if, and only if, $[\lambda(t_i{}^{x_{k1}})] = [\lambda(t_j{}^{x_{k2}})] = x_k$.

Proof. Suppose $[\lambda(t_i{}^{x_{k1}})] = [\lambda(t_j{}^{x_{k2}})] = x_k$. Consider some $\lambda(c_p) \not\sim \lambda(c_q)$. Let s be the smallest integer such that $c_{ps} \wedge c_{qs} = \phi$ and suppose with no loss in generality, $c_{ps} = 1$. From the construction of T^o and T^1 either $c_p \subseteq t_i^1$ and $c_q \subseteq t_i^o$ or $c_p \subseteq t_j^1$ and $c_q \subseteq t_j^o$. Whichever, it follows that either $[\lambda(c_p)] = x_k$ or $[\lambda(c_q)] = x_k$.

Suppose $[\lambda(c_p)] = x_k$ or $[\lambda(c_q)] = x_k$ for all $\lambda(c_p) \not\sim \lambda(c_q)$. Further suppose $[\lambda(t_i{}^{x_{k1}})] \neq x_k$. There exists some $c_\ell \subseteq t_i{}^{x_{k1}}$, $c_m \subseteq t_i{}^{x_{k1}}$ such that

$[\lambda(c_\ell)] \neq x_k$ and $\lambda(c_\ell) \not\sim \lambda(c_m)$. Clearly, $[\lambda(c_m)] \neq x_k$ as $c_{mi} = \bar{x}_{k1}$. This is a contradiction. A similar contradiction arises if we assume $[\lambda(t_j^{x_{k2}})] \neq x_k$, hence, $[\lambda(t_i^{x_{k1}})] = [\lambda(t_j^{x_{k2}})] = x_k$. QED.

Theorem 4.6. Given x_k and p, $\lambda(c_\ell)_p \wedge \lambda(c_m)_p = x_{kp}$ for all $\lambda(c_\ell) \not\sim \lambda(c_m)$ such that $\lambda(c_\ell)_p \wedge \lambda(c_m)_p \neq \phi$ if, and only if, $p = 1$ and $u_{ji} = x_{k1}$ or $p = 2$ and $u_{ij} = x_{k2}$.

Proof. Consider $p = 1$. Suppose $u_{ji} = x_{k1}$. Consider $\lambda(c_\ell) \neq \lambda(c_m)$ such that $\lambda(c_\ell)_1 \wedge \lambda(c_m)_1 \neq \phi$. From the construction of U, $c_\ell \wedge c_m \subseteq u_j$ and thus $\lambda(c_\ell)_1 \wedge \lambda(c_m)_1 \subseteq u_{ji}$. Clearly, $\lambda(c_\ell)_1 \wedge \lambda(c_m)_1 = x_{k1}$.

Suppose $\lambda(c_\ell)_1 \wedge \lambda(c_m)_1 = x_{k1}$ for all $\lambda(c_\ell) \not\sim \lambda(c_m)$ such that $\lambda(c_\ell)_1 \wedge \lambda(c_m)_1 \neq \phi$. Suppose $u_{ji} \neq x_{k1}$. From the construction of U there is some c_r, c_s such that $c_r \wedge c_s \subseteq u_j$; and $\lambda(c_r)_1 \wedge \lambda(c_s)_1 \neq x_{k1}$, $\lambda(c_r)_1 \wedge \lambda(c_s)_1 \neq \phi$.

This is a contradiction and it follows that $\lambda(c_\ell)_1 \wedge \lambda(c_m)_1 = x_{k1}$ for all $\lambda(c_\ell) \not\sim \lambda(c_m)$ such that $\lambda(c_\ell)_1 \wedge \lambda(c_m)_1 \neq \phi$ implies $u_{ji} = x_{ki}$.

The proof is analogous for $p = 2$. QED.

Theorem 4.7. $00 \not\sim 11$ if, and only if, $v_{ij} \supseteq 1$ or $v_{ji} \supseteq 1$.

Proof. Suppose $v_{ij} \supseteq 1$. Clearly there exists some $\lambda(c_p) \not\sim \lambda(c_q)$ such that $\lambda(c_p)_1 \wedge \lambda(c_q)_1 = \phi$ and, assuming with no loss of generality that $c_{p1} = 1$,

$\lambda(c_p)_2 \zeta \lambda(c_q)_2 = 1$. From the definition of ζ, $\lambda(c_p)_2 \supseteq 1$ and $\lambda(c_q)_2 \supseteq 0$. It follows that $11 \varepsilon [\lambda(c_p)]$ and $00 \varepsilon [\lambda c_q)]$ and $00 \not\sim 11$. Similarly, $v_{ji} = 1$ implies $00 \not\sim 11$.

Suppose $11 \not\sim 00$. Clearly, there exists some $\lambda(c_p) \not\sim \lambda(c_q)$ such that $11 \varepsilon [\lambda(c_p)]$ and $00 \varepsilon [\lambda(c_q)]$. Let s be the smallest integer such that $c_{ps} \wedge c_{qs} = \phi$. Clearly, $s = i$ or $s = j$. $\lambda(c_p) \zeta \lambda(c_q) \supseteq 11$. It follows from the construction of V that $v_{ij} \supseteq 1$ or $v_{ji} \supseteq 1$ as $c_p \zeta c_q \subseteq v_s$. QED.

Theorem 4.8. $01 \not\sim 10$ if, and only if, $v_{ij} \supseteq 0$ or $v_{ji} \supseteq 0$.

Proof. The proof is analogous to the proof of theorem 4.7.

Theorems 4.1 through 4.4 establish simple necessary and sufficient conditions for the existence of certain two-place decompositions of two-valued functions. Lemmata 4.1 and 4.2 and theorems 4.5 through 4.8 show that these conditions can be tested by examining T^0, T^1, U and V as constructed in algorithm 4.1. The principal advantage of this approach is that the construction of T^0, T^1, U and V requires each pair of cubes in the definition of f be compared once. If the general approach developed for many-valued functions is employed each pair of cubes must be compared for each pair of variables.

The following table presents several decomposition

tests in terms of T^o, T^1, U and V. For each α_1 and α_2 indicated g exists if, and only if, the accompanying test is true.

Decomposition Tests

α_1	α_2	test
$\delta(\alpha_1) = 0$	$\alpha_2 = \beta[11]$	$t^1_{ij} = t^1_{ji} = 1$
$\delta(\alpha_1) = 0$	$\alpha_2 = \beta[10]$	$t^1_{ij} = 0$ and $t^o_{ji} = 1$
$\delta(\alpha_1) = 0$	$\alpha_2 = \beta[01]$	$t^o_{ij} = 1$ and $t^1_{ji} = 0$
$\delta(\alpha_1) = 0$	$\alpha_2 = \beta[00]$	$t^o_{ij} = t^o_{ji} = 0$
$\delta(\alpha_1) = 0$	equivalence or nonequivalence	$v_{ij} \cup v_{ji} = \phi$
$\alpha_1 = a_i$ or $\alpha_1 = \bar{a}_i$	$\alpha_2 = \beta[0-]$	$u_{ji} = 0$
$\alpha_1 = a_i$ or $\alpha_1 = \bar{a}_i$	$\alpha_2 = \beta[1-]$	$u_{ji} = 1$
$\alpha_1 = a_j$ or $\alpha_1 = \bar{a}_j$	$\alpha_2 = \beta[-0]$	$u_{ij} = 0$
$\alpha_1 = a_j$ or $\alpha_1 = \bar{a}_j$	$\alpha_2 = \beta[-1]$	$u_{ij} = 1$
$\alpha_1 = \beta[00]$	$\alpha_2 = \beta[11]$	$v_{ij} \cup v_{ji} = 1$
$\alpha_1 = \beta[01]$	$\alpha_2 = \beta[10]$	$v_{ij} \cup v_{ji} = 0$

Note: A dash (-) indicates a choice of 0 or 1.

Simple nondisjunctive decompositions with α_2 the equivalence or nonequivalence are not considered since it is easily shown such a decomposition must be trivial. For any complex decomposition other than the two considered, there is always a simple nondisjunctive decomposition. Since the latter use less hardware they will be chosen in practice. We have thus only included

the complex decompositions which can possibly occur as the only choice.

4. AN EXAMPLE

An example will illustrate the efficiency of these tests. Consider the set of cubes C = {xx11,101x, 011x,11x0,00x0,xx0x} and let $f(a_1,a_2,a_3,a_4)$ be such that f[xx11] = f[101x] = f[011x] = 1 and f[11x0] = f[00x0] = f[xx0x] = 0.

Initially we set $t^o_\ell = t^1_\ell = u_\ell = v_\ell = \phi$ for all $1 \supseteq \ell \supseteq 4$. Consider c_1 = xx11 and c_4 = 11x0. $c_1 \wedge c_4 = 111\phi$ and $\Delta(c_1,c_4) = 1$. From algorithm 5.1 we get

$$t^o_4 = \phi \cup 11x0 = 11x0$$

$$t^1_4 = \phi \cup xx11 = xx11$$

$$u_4 = \phi \cup (xx11 \wedge 11x0) = 111\phi$$

$$v_4 = \phi \cup (xx11 \zeta 11x0) = 0011 .$$

Now consider c_1 = xx11 and c_5 = 00x0. $c_1 \wedge c_5 = 001\phi$ and $\Delta(c_1,c_5) = 1$. From algorithm 5.1 we get

$$t^o_4 = 11x0 \cup 00x0 = xxx0$$

$$t^1_4 = xx11 \cup xx11 = xx11$$

$$u_4 = 111\phi \cup (xx11 \wedge 00x0) = xx1\phi$$

$$v_4 = 0011 \cup (xx11 \zeta 00x0) = xx11.$$

Proceeding through all c_p, c_q such that $f[c_p] \neq f[c_q]$ we finally get

$$T^o = \{0xxx,x0xx,xx0x,xxx0\}$$

$$T^1 = \{1xxx,x1xx,xx1x,xx11\}$$

$$U = \{\phi x10,x\phi 10,xx\phi x,xx1\phi\}$$

$V = \{1\phi xx, \phi 10x, xx1x, xx11\}$

Applying the decomposition tests for each $a_i a_j$ pair $1 \leq i \leq 4$, $1 \leq j \leq 4$ we find g exists for the following choices of α_1 and α_2:

i)	$\alpha_1(a_1,a_2) = 0$	$\alpha_2(a_1,a_2)$ = equivalence or nonequivalence
ii)	$\alpha_1(a_1,a_3) = a_3$ or $\bar{a}_3$	$\alpha_2(a_1,a_3) = \beta[-1]$
iii)	$\alpha_1(a_1,a_4) = a_4$ or $\bar{a}_4$	$\alpha_2(a_1,a_4) = \beta[-0]$
iv)	$\alpha_1(a_2,a_3) = a_3$ or $\bar{a}_3$	$\alpha_2(a_2,a_3) = \beta[-1]$
v)	$\alpha_1(a_2,a_4) = a_4$ or $\bar{a}_4$	$\alpha_2(a_2,a_4) = \beta[-0]$
vi)	$\alpha_1(a_3,a_4) = a_3$ or $\bar{a}_3$	$\alpha_2(a_3,a_4) = \beta[1-]$

5. REMARKS

Treating two-valued functions separately has led to a much simpler determination technique than the general algorithm discussed in chapter 3. The representation of cubes and the construction and examination of T^0, T^1, U and V are quite suited to a computer. A major advantage is the minimization of the hardware cost of realizing each decomposition. This is accomplished by applying the decomposition tests in order according to the hardware elements being considered. This is superior to the approach in chapter 3 since universal decision elements are seldom used in two-valued logic.

The sets T^0, T^1 and U correspond to the decomposition tables due to Barnard and Holman [3]. The addition of the set V allows us to consider complex

decompositions as well as decompositions involving the equivalence or nonequivalence. We shall find these types of decomposition indispensable in the practical design environment.

An extension of the cube notation to three-valued functions has been considered in [45]. For three values the idea is reasonable. For four values, however, sixteen symbols are required and the set notation of chapter 3 is more descriptive and just as easily implemented on the computer.

The techniques developed in this chapter are strictly two-valued techniques. For more than two values, too much information is required for it to be represented in a form as compact and easily constructed as T^0, T^1, U and V. We will find in the next chapter that these techniques lead to an efficient two-valued synthesis algorithm. In addition, they will be used to synthesize two-valued multiple-output switching circuits, a problem which has previously been computationally too complex for decomposition techniques.

CHAPTER 5

Synthesis of Single-Output Circuits

1. INTRODUCTION

In chapters 3 and 4, techniques were developed for identifying two-place decompositions of many-valued and two-valued functions. We now consider the application of these techniques to the synthesis of switching circuits. In this chapter, we consider single-output combinational circuits. Multiple-output circuits are considered in the following chapter. Sequential circuits are considered briefly in chapter 7.

A single-output combinational circuit is one whose output depends solely on the current values of its inputs. It contains no memory or feedback loops. The behaviour of this kind of circuit is completely defined by a single two-valued or many-valued function. The synthesis problem is to construct a switching circuit to realize a given function using a certain set of primitive switching elements.

A synthesis algorithm based on two-place decomposition will be developed. This algorithm constructs a sequence of decompositions. The first of these decompositions applies to the initial function. Each subsequent decomposition applies to the image of its predecessor. The image of the final decomposition is a two-place function. This sequence of decompositions specifies a realization of the

given function as a composition of two-place functions. These two-place functions are straightforward to implement in hardware thus yielding an implementation of the given function.

The algorithm is based on the algorithm due to Roth and Karp [59] and is a continuation of the work presented by the author in [36]. It is quite simple since it constructs a single sequence of decompositions by the repeated application of heuristic selection criteria which at each stage choose the 'best' decomposition. No search techniques such as 'back-tracking' or 'look-ahead' are employed. The algorithm is consequently very efficient in terms of both execution time and storage.

Implementation details are quite different for many-valued and two-valued functions since different techniques are used to identify two-place decompositions and to find the images of the chosen decompositions. The structure of the algorithm and, in particular, the heuristic selection criteria are the same in both cases. The algorithm has been implemented on the computer for both cases. Several sample problems have been solved. A number of these are presented in this chapter and compared to solutions found using alternative techniques. Since heuristics are used, empirical tests are the only means of evaluating the algorithm.

2. ROTH AND KARP'S ALGORITHM

Roth and Karp's decomposition algorithm [59] is a systematic procedure for the design of single-output two-valued combinational switching circuits. The distinquishing feature of this algorithm is that it requires no restrictive assumptions about the primitive switching elements used, their costs or the manner in which they may be interconnected except that feedback loops are not permitted. This flexibility is the principal feature of synthesis techniques based on decomposition and is in contrast with previous techniques which deal almost exclusively with special problems such as the design of two-level circuits. The drawback to Roth and Karp's algorithm is the extensive computation required. The algorithm developed later in the chapter retains much of the scope and flexibility of Roth and Karp's algorithm while greatly reducing the required computation.

Roth and Karp considered decompositions of the form

$$f(A) = g(\alpha_1(A_\lambda), \alpha_2(A_\lambda), \ldots, \alpha_t(A_\lambda), A_\mu) \qquad (1)$$

where $A_\lambda \cup A_\mu = A$, and where $f, g, \alpha_1, \alpha_2, \ldots, \alpha_t$ are all two-valued. The sets A_λ and A_μ need not be disjoint. The decompositions considered are restricted to those where the α_i are taken from a set of primitive functions specified as part of the problem. In the implementation due to Karp *et al* [60] these primitive functions were

taken to be the vertex functions.

At the beginning of the synthesis procedure nothing is known of the switching circuit to realize f and it can only be considered as a black box with some number of inputs and a single output.

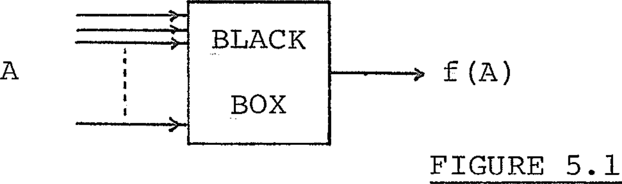


FIGURE 5.1

A decomposition of the form (1) represents a replacement of this single box by a composition of some number of boxes each realizing a primitive function α_i and another black box implementing g.

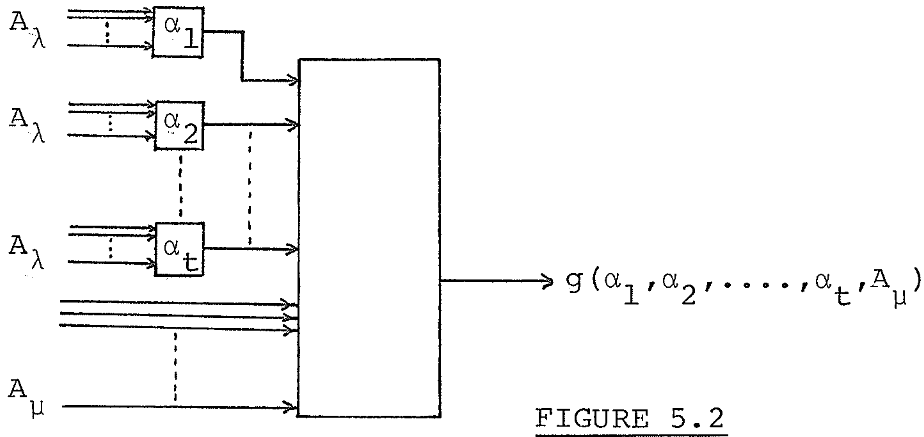


FIGURE 5.2

A second decomposition can then be found by examining the function g. The black box implementing g can then be replaced by a composition of switching elements and yet another black box. This procedure is repeated until a final decomposition is found which expresses the function

being considered at that step entirely in terms of primitive switching elements. The result is a realization of the initial function in terms of primitive switching elements. This switching circuit is completely specified by the sequence of decompositions.

The realization of a function is not unique and there are several sequences of decompositions representing realizations of a given function. Roth and Karp's algorithm is a cost bounded search of all possible sequences and hence all possible realizations employing the prescribed set of primitive functions. The cost of a circuit is taken as the sum of the costs of the switching elements comprising the circuits.

The search process may be viewed as an ordered tree search. The nodes of this tree correspond to functions. The root is the initial function to be realized. Each of the other nodes is the image of some decomposition. The leaves correspond to primitive switching functions. The branches correspond to decompositions. Paths in the tree represent sequences of decompositions each operating on the image of its predecessor. Paths from the root to a leaf represent switching circuits implementing the given function.

The search is carried out in a manner analogous to the search employed in the graph colouring algorithm described in chapter 3. Initially one path from the

root to a leaf is traversed. This gives both an initial realization and a first approximation to the minimal cost of a realization. Back-tracking techniques are then employed to examine other paths in the tree. The traversal of a path is terminated when either:

a) the cost of the elements required in the partial solution specified at any point in this path exceeds the known minmum cost of a total realization,

or b) a leaf is reached in which case a new realization with a lower cost has been found.

The search terminates when all possible paths have been considered.

Two paths in the tree may specify the same sequence of decompositions in two different orders. Techniques are incorporated into the search so that each unique sequence is examined only once. Techniques are also employed to avoid decompositions resulting in <u>cycling</u> a phenomenon which produces an α_i identical to

a) a constant,

b) an input to α_i or its complement,

c) an α_j produced by a previous decomposition or its complement.

It is easily shown that a circuit employing cyclic elements is never minimal.

As Karp et al [60] have observed the search can be greatly reduced by employing selection criteria to bias the initial path examined towards a 'good' solution. Unfortunately, the actual criteria used were not specified.

The above approach has a number of computational drawbacks. First, as was found in chapter 2, the identification of decompositions of the form (1) is very difficult except in highly specialized cases such as when the α_i are restricted to the vertex functions. This is particularly true if the technique is to be extended to many-valued functions. For a given function of n arguments there are 2^n-n-1 choices for A_λ and for each of these choices several possible A_μ. It is interesting to note that in both [59] and [60] the examples given employ two-input gates except for a single three-input gate. No indication was given whether computational problems excluded the use of multiple-input gates or whether they simply did not arise. The former appears more likely since many circuits employing multiple-input gates are known.

The identification of decompositions, determining the images of decompositions and directing the search involve a great deal of computation. The latter process requires keeping track of the functions in the sequence being considered as well as their decompositions. A great deal of this computation may be wasted. Roth and

Karp give an example where a circuit found in a few seconds was not improved in almost an hour of further searching.

3. TWO-PLACE DECOMPOSITION AND THE SYNTHESIS PROBLEM

The study of the identification of decompositions in chapters 2 and 3 led to the consideration of the special case of two-place decompositions. Efficient techniques for this special case were developed in chapters 3 and 4. We now consider how these techniques can be employed to advantage in a synthesis procedure.

The two-place decomposition techniques could be incorporated into Roth and Karp's algorithm. The resulting process would determine a minimal cost realization constructed of two-input elements and could be applied to both two-valued and many-valued functions. In theory, the modified algorithm is, in the two-valued case, less general than Roth and Karp's algorithm. In practice, however, Roth and Karp's algorithm has only been used to produce circuits employing two-input elements and these elements have been restricted to those which implement vertex functions. The two-valued two-place techniques in chapter 4 allow the exclusive-OR function and are much more efficient than applying Roth and Karp's general technique to two-place decomposition. Simply incorporating the two-place techniques would thus result in an improvement in both the scope and efficiency of Roth

and Karp's algorithm.

Barnard and Holman [3] found that a search for a minimal solution is prohibitive even when efficient techniques for identifying decompositions are employed. Using their tabular techniques, described in chapter 4, the circuit in figure 5.3 was found in 68 seconds on a KDF9 computer. The initial result was found in 1.6 seconds. This long execution time for a simple problem is not promising, especially if a multiple-output algorithm is to be developed where the size of the decomposirion tree will grow very much larger than in the single-output case. There is also the disadvantage that a search algorithm is rather complex to implement and requires a large amount of storage.

Roth and Karp [59] have suggested a number of possible solutions. A partial search could be made and a 'good' rather than a minimal result accepted. Heuristic criteria for directing the choice of the first circuit produced i.e. the first path followed in the decomposition tree, can reduce the length of the search by generating a good initial result thereby placing a tighter bound on the circuits considered. Roth and Karp found empirically that a great deal of the search is often spent in minimizing the realization of a function of three arguments encountered during construction of a realization. This could be avoided by using a table of the optimum reali-

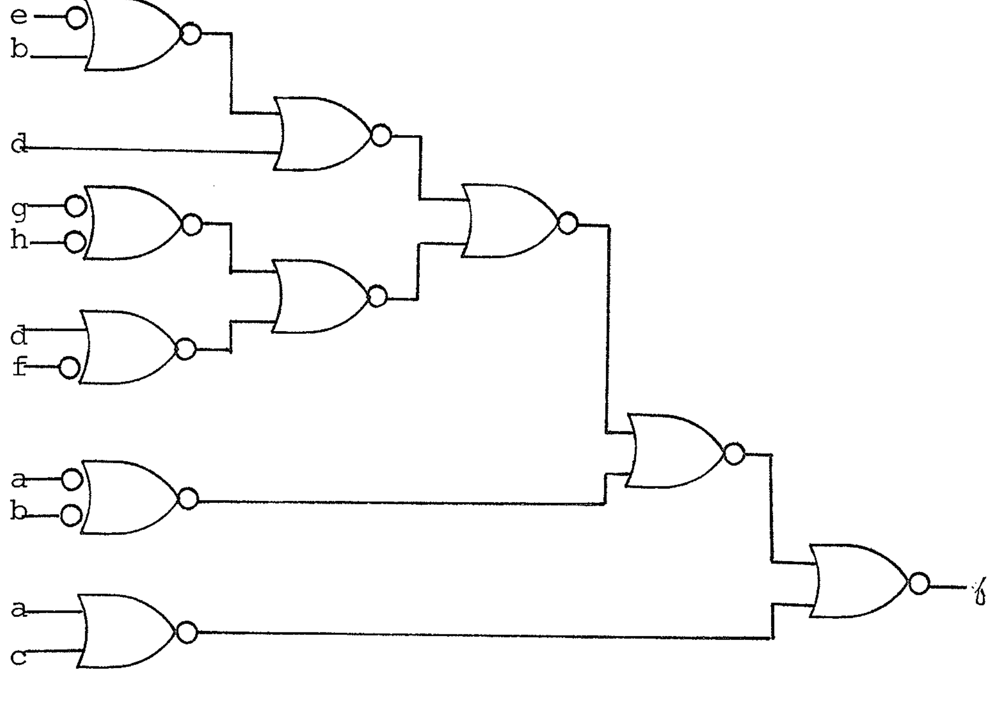


FIGURE 5.3

zations of all three argument functions. This is of course not practical in the many-valued case.

The algorithm below takes yet another approach. A single sequence of decompositions is constructed using no 'back-tracking' or 'look-ahead' techniques. At each stage the decompositions of the function being considered are examined and a 'best' decomposition is chosen according to a set of heuristic selection criteria. The chosen decomposition is implemented and the process is applied to the resulting image. A major objective of this work was to develop a simple, very fast method which could be extended to the multiple-output case. Empirical test will show that the particular selection criteria presented yield very good results which are comparable to those produced by more complex procedures.

4. A SYNTHESIS ALGORITHM

The algorithm below applies to both many-valued and two-valued functions. We assume two-place many-valued functions are to be realized using a universal decision element. Recall that this assumption was also made in chapter 3 in developing techniques for the identification of two-place many-valued decompositions. The principal result of this assumption is that the cost of realizing a two-place r-valued function is fixed by r and not the function being realized. The synthesis algorithm makes no decision based on the particular functions encountered.

It is anticipated that two-place two-valued functions are to be realized by more traditional methods and not by universal decision elements. The cost of realizing the function in this case does depend on the function. This cost will, however, be ignored by the synthesis algorithm. There are a number of reasons for this. The vast majority of two-valued switching circuits are realized using integrated circuits. In this technology the number of interconnections not the cost of the elements is the principal guideline for a 'good' circuit. It is thus the number of two-place functions used in a realization which is important, Extensive testing [36] has shown that selection criteria incorporating the cost of the two-place functions in general produce no better results than the methods below. Finally, ignoring the cost criteria results in a simpler and more efficient algorithm which is easier to implement.

This synthesis algorithm constructs a sequence of two-place decompositions. The first decomposition in this sequence operates on the given function. Each of the remaining decompositions operates on the image of its predecessor. The image of the final decomposition in this sequence is a two-place function. At each step the algorithm examines the two-place decompositions of the function being considered and selects one according to a number of heuristic rules. These rules attempt

to reduce the number of two-place functions required to realize the initial function as well as the number of levels in the corresponding switching circuit.

Two-place decompositions involving one-place functions are ignored since universal decision elements are to be used in the many-valued case which, as described in chapter 3, makes one-place functions superfluous. One-place functions never arise in the two-valued case. Three types of two-place decomposition are considered:

SIMPLE DISJUNCTIVE

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

SIMPLE NONDISJUNCTIVE

$$f(a_1,a_2,\ldots,a_n) = g(\alpha(a_i,a_j),a_1,a_2,\ldots\ldots,a_{j-1},a_{j+1},\ldots,a_n)$$

COMPLEX DISJUNCTIVE

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_1(a_i,a_j),\alpha_2(a_i,a_j),a_1,a_2,\ldots\ldots,a_{i-1},a_{i+1},\ldots,a_{j-1},a_{j+1},\ldots,a_n)$$

When one exists a simple disjunctive decomposition is chosen before either of the other two types. If no simple disjunctive decompositions exist, a simple nondisjunctive decomposition is chosen. A complex disjunctive decomposition is chosen only when no other choice is available. This order of preference choses a decomposition involving the least number of true two-place functions and the least number of arguments to the image

of the decomposition. Note that this order of preference corresponds to the order of preference in assigning the type of a two-place decomposition for a particular a_i, a_j pair.

Often there are several decompositions of the same type. In this case, the algorithm chooses the type as above, and then examines the arguments of each of the decompositions of this type, determining for each decomposition which argument has come through the most levels. The decomposition for which this value is minimal is selected. If two decompositions of the selected type have the same minimal value, the first to be generated is selected. Decompositions are identified by examining the arguments in the order $(a_1,a_2),(a_1,a_3),\ldots$ $\ldots,(a_1,a_n),(a_2,a_3),\ldots,(a_{n-1},a_n)$. New arguments to the image of the decomposition are added following a_n.

In implementing the algorithm, the identification and selection processes are actually combined into a single step and are not performed in sequence as suggested above. The three types of two-place decompositions, simple disjunctive, simple nondisjunctive and complex disjunctive, are assigned relative costs of 1, 2, and 3. As each decomposition is found, this cost is combined with the gating level of the decomposition i.e. the maximum number of levels through which one of the arguments of the decomposition has passed. The result

termed the figure of compexity is defined by

$$\text{figure of complexity} = \text{cost} \times 1000 + \text{level of gating}$$

This value combines the two selection criteria into one. It is easily verified that the decomposition chosen by the selection criteria described above has a minimal figure of complexity. If two decompositions have the minimal figure of complexity the first to be generated is chosen. The value 1000 was chosen so that the level of gating does not intefere with the cost. By proper use of these figures of complexity, only two decompositions need be recorded at any one time, the decomposition currently being considered and the decomposition with the minimal figure of complexity known to date. A flowchart of the algorithm as implemented is given in figure 5.4.

5. THE IDENTIFICATION OF TWO-PLACE DECOMPOSITIONS

Techniques for the identification of two-place decompositions were developed in chapters 3 and 4. We now outline how these techniques are employed in the synthesis algorithm. Many-valued functions are considered first.

For a many-valued function each pair of variables is examined seperately. Consider an r-valued function $f(a_1,a_2,\ldots,a_n)$ and pair of variables a_i,a_j. Algorithm 3.1 is applied to find the compatible and incompatible

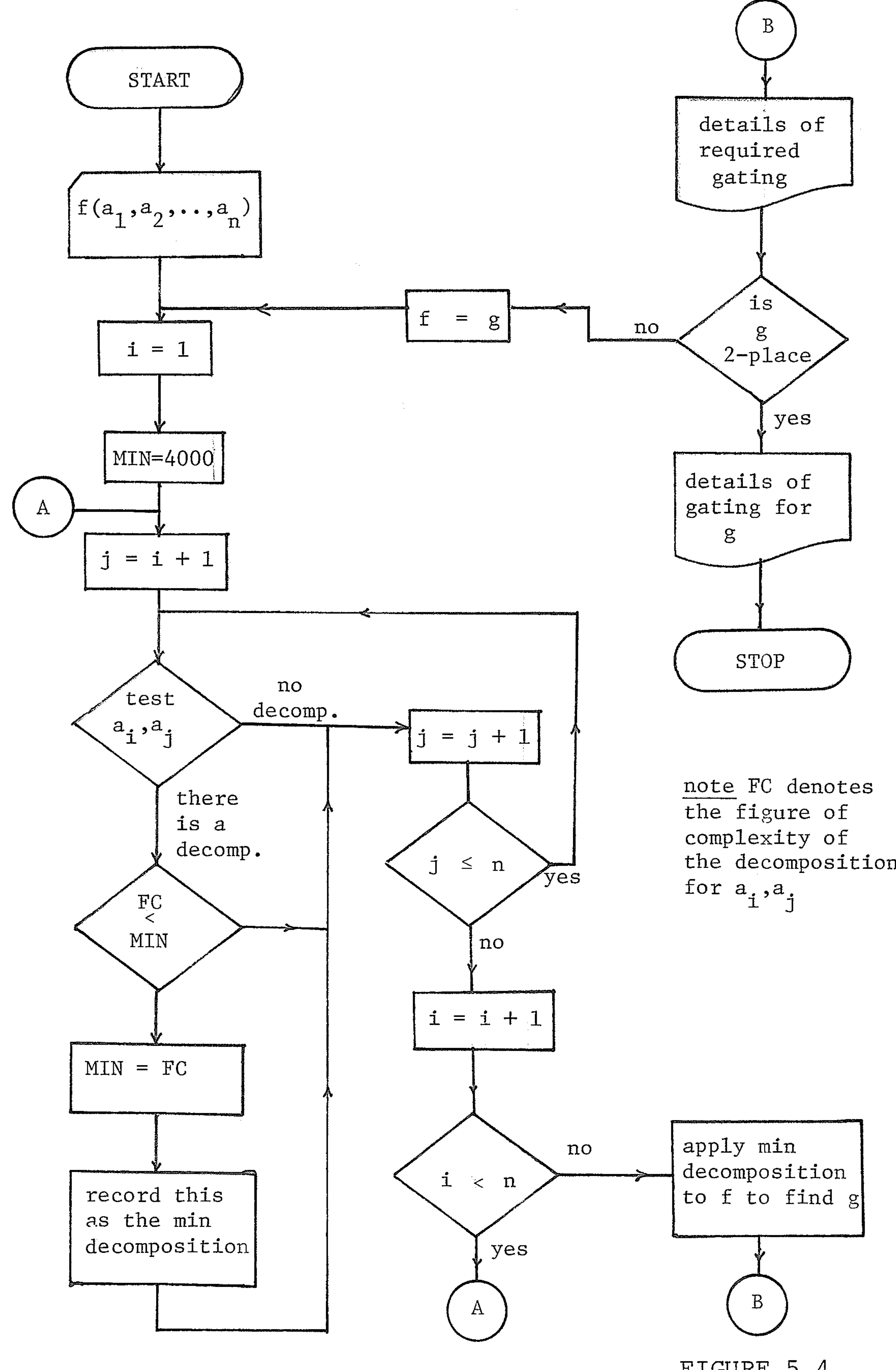


FIGURE 5.4

pairs of X, the set of assignments to (a_i, a_j) in the definition of f. The incompatibility graph G representing these relations is formed and a minimal colouring of G is found using the algorithm described in chapter 3. If $\chi(G) = |X|$ there is no nontrivial two-place decomposition. If $\chi(G) \leq r$ there is a simple disjunctive decomposition where $\alpha_1(a_i, a_j)$ is 0 and $\alpha_2(a_i, a_j)$ is found as in the proof of theorem 3.2. If $r < \chi(G) < p$ the colouring is stored for possible use in determining a complex disjunctive decomposition.

If no simple disjunctive decomposition is found the simple nondisjunctive case is examined as follows. If there is some $x_s \sim x_t$, $x_s, x_t \varepsilon X$ such that x_s and x_t agree in the coordinate corresponding to a_i, there is a simple nondisjunctive decomposition with $\alpha_1(a_i, a_j) = a_i$. $\tilde{G}$ is formed from G, a minimal colouring of $\tilde{G}$ is determined and from this colouring $\alpha_2(a_i, a_j)$ is found. This process is described in section 8 of chapter 3. A similar sequence is used to determine a simple nondisjunctive decomposition with $\alpha_1(a_j) = a_j$. When simple nondisjunctive decompositions exist both for $\alpha_1(a_i, a_j) = a_i$ and for $\alpha_1(a_i, a_j) = a_j$, the one for which α_2 has the smaller range is chosen. If the range of α_2 is the same in both cases the decomposition with $\alpha_1(a_i, a_j) = a_i$ is chosen.

If no other type of decomposition is possible a complex disjunctive decomposition is found from the colouring of G as described in section 9 of chapter 3.

The above procedure is not applied in full for each pair of variables. Once a simple disjunctive decomposition has been found, no further pairs are examined for simple nondisjunctive or complex decompositions since the selection criteria prefer a simple disjunctive decomposition. Similarly, detecting a simple nondisjunctive decomposition eliminates the need to look for any further complex decompositions. These observations will often greatly reduce the required computation.

The two-valued case is easier, much of the computation being done in parallel for all pairs of variables. Algorithm 4.1 is used to determine the sets T^0,T^1,U and V from a cube definition of the function being considered. This construction requires each pair of cubes be examined once. Once T^0,T^1,U and V are known, the possible decompositions are determined by applying the decomposition tests of section 3 of chapter 4, for each pair of variables. These tests simply involve an examination of certain elements of T^0,T^1,U and V and require no complex computation. They are extremely fast. For each a_i,a_j pair the 'best' decomposition is the one corresponding to the satisfied test which appears highest in the list.

As Roth and Karp noted decompositions introducing cycling should be avoided. In two-place decomposition cycling conditions are easily detected since the range

and domain of $\alpha=(\alpha_1,\alpha_2)$ have the same number of elements. This can only occur for simple nondisjunctive or complex disjunctive decompositions and is easliy avoided by comparing the range and domain of each α encountered.

6. THE IMAGE OF A TWO-PLACE DECOMPOSITION

Consider a two-place decomposition

$$f(a_1,a_2,\ldots,a_n) = g(\alpha_1(a_i,a_j),\alpha_2(a_i,a_j),a_1,a_2,\ldots \\ \ldots,a_{i-1},a_{i+1},\ldots,a_{j-1},a_{j+1},\ldots,a_n).$$

The function g is the image of the decomposition. Once α_1 and α_2 are chosen, the value of g is fixed for every assignment to its arguments for which it need be defined. g may have a number of 'don't-care' conditions. These are simply ignored. In the two-valued case the tabular procedure developed in [36] is used. The many-valued case is handled as follows.

For a simple disjunctive decomposition a matrix definition of g is found from the matrix definition of f by a straightforward substitution process. For each row of the matrix defining f, form a set of values assumed by α_2 for the assignment to a_i,a_j specified by this row. This set is added to the row and the sets corresponding to a_i and a_j are removed. Applying this process to all the rows in the matrix results in a matrix defining g.

In the simple nondisjunctive case with a_k, k=i or k=j, the common argument to g and α_2, the matrix defining

f must be expanded so that each set specifying values for a_k contains a single element. Rows where this is not the case must be split into some number of rows each satifying this condition. This expansion is necessary to avoid inconsistencies in the image. Once the expansion is made, the matrix defining g is found as described above except the sets corresponding to a_k are retained. For complex disjunctive decompositions the process is the same as for simple disjunctive decompositions except that sets are added for both α_1 and α_2.

Once g is found it is useful to compress its definition. This reduces the computation required to find the decompositions of this function. We remove cubes or rows which define input assignments which are defined by one other cube or row. More elaborate procedures are possible, but the additional computation does not, in general,result in a significantly more compact definition. The reduction we have chosen is straightforward and can be performed as the image of the decomposition is found.

7. COMPUTER IMPLEMENTATION

Since the synthesis algorithm is based on heuristic selection criteria, it must be evaluated empirically. Two programs, one for many-valued problems and one for two-valued problems, have been written in the FORTRAN IV language [24] and implemented on an IBM 370/158

computer [25] using the FORTRAN-H compiler [23]. Before examining the results produced by these programs we shall consider a few of the more interesting techniques used in implementing the algorithm.

Many-valued functions are represented as a matrix of sets. In our implementation each of these sets is represented by a bit string stored in one byte (8 bits). The program will thus accomodate functions in up to 8 values. At present the matrix may have up to 100 rows and 16 columns. The latter limits are easily modified. Extending the program to functions in more than eight values would require extensive reprogramming.

The synthesis algorithm requires the following operations on the sets representing a many-valued function:

a) determine if two sets intersect,

b) determine if a given value is in a set,

c) remove a value from a set.

The first operation is performed by taking the logical AND of the bytes representing the two sets. If the resulting byte is all zeros, the sets are disjoint, otherwise, they intersect. The second operation is performed by taking the logical AND of the byte representing the set with a byte containing a single one in the bit position corresponding to the value in question. If the resulting byte is all zeros the set does not contain the value. The third operation is performed

by replacing the byte representing the set by the logical AND of that byte with a byte containing all ones except for a zero in the position corresponding to the value of interest. The logical AND and testing if a byte is all zeros are basic machine language operations accessible to FORTRAN via assembler language [26] subroutines.

A two-valued function is represented by a set of cubes. Each cube is represented by a bit string stored in a full word (32 bits). The symbols appearing in a cube are each represented by a pair of bits according to the following table:

symbol	representation
ϕ	0 0
0	0 1
1	1 0
x	1 1

Using a single word for each cube accomodates functions in up to 15 variables since the two other bits are required to represent the value of the function for that cube.

When applying the synthesis algorithm to two-valued functions the operators $\cup$,$\wedge$,ξ and $\cap$ as defined in chapter 4 are required. $\cup$ is the logical OR of the words representing the two cubes. $\wedge$ is the logical AND. $\cap$ is obtained from the $\wedge$ result by examining the bits in the result in pairs. ξ is more complex requiring five logical

operations and two shifts of one bit. Two full word bit masks are also used. Let AND denote logical AND and let OR denote logical OR. Let T0 and T1 be the words representing the two cubes being considered and let M1 and M2 be two 32-bit masks where

M1 = 1010....10

M2 = 0101....01

Let SHR denote a right shift of one bit where the right bit is lost and the left bit becomes 0. Let SHL denote the analogous left shift of one bit. The result of applying ξ to the cubes represented by T0 and T1 can be shown to be represented by

OR(AND(M1,AND(T1,SHL(T0))),AND(M2,AND(T1,SHR(T0))).

Implementing $\cap$ requires the operation of identifying the value of a particular coordinate of a cube. This is most easily accomplished by shifting the bit string right the required number of positions and to then form the logical AND of this result with the bit string 00....011. Altering the value of a coordinate is accomplished by removal of the old value and insertion of the new, both of which are accomplished in a manner similar to the methods described for the many-valued case.

The above techniques are those which are peculiar to this problem. The rest of the implementation uses well-known programming methods.

8. TWO-VALUED EXAMPLES

Three criteria will be used to evaluate circuits. They are: the number of gates in the circuit; the gating level i.e. the maximum number of gates which a signal must pass through between an input and the output; the number of connections in the circuit. The majority of two-valued switching circuits are now realized as single integrated chips or by interconnecting some number of chips. Each of these chips contains several gates. The first criterion is, however still useful as an estimate of the complexity of the circuit. The second criterion is an estimate of the timing delay of the circuit. An exact timing would depend on the actual hardware used. In integrated circuit technology the number of connections is the most important criterion with respect to the hardware cost. In evaluating our circuits the number of gate inputs will be used as the number of connections. We will refer to this number as the cost of the circuit.

Inverters on inputs and outputs are not considered in any of the three evaluation criteria. In many practical problems, the inputs are available in both true and inverted form and an inverted output is acceptable. This is particularly true when the circuit being synthesized is one part of a large digital system. The inputs may be the outputs of other circuits and may be available in true or inverted form or both. An output to be used

in a later part of the system may be perfectly acceptable in inverted form. There is no point in attempting to evaluate the inputs and outputs of a circuit outside the context within which it is to be used.

EXAMPLE 1.

The first example is a validity checker for the 'two out of five' code. The output of this circuit is one whenever exactly two of the inputs are one. Our result is shown in figure 5.5. The program produced the circuit in 1.52 seconds. Roth and Karp's solution [59] is shown in figure 5.6. The circuit used in the IBM 7090 computer is shown in figure 5.7.

Figure 5.8 shows the program result modified to use AND and OR gates. This allows it to be more closely compared with Roth and Karp's result. This modification was done by hand, but is equivalent to modifying the program to handle these types of gates. Our result and Roth and Karp's result are identical up to and including the gates labeled α, β and γ. From that point our result is simpler because of the use of an EXOR gate which was not allowed in Roth and Karp's algorithm. Figure 5.6 is the most complex published result produced by Roth and Karp's algorithm. Our result uses three fewer gates, two fewer levels and has a cost of 24 as opposed to 29.

Of particular interest in this example is the number of complex disjunctive decompositions. Barnard and Holman's

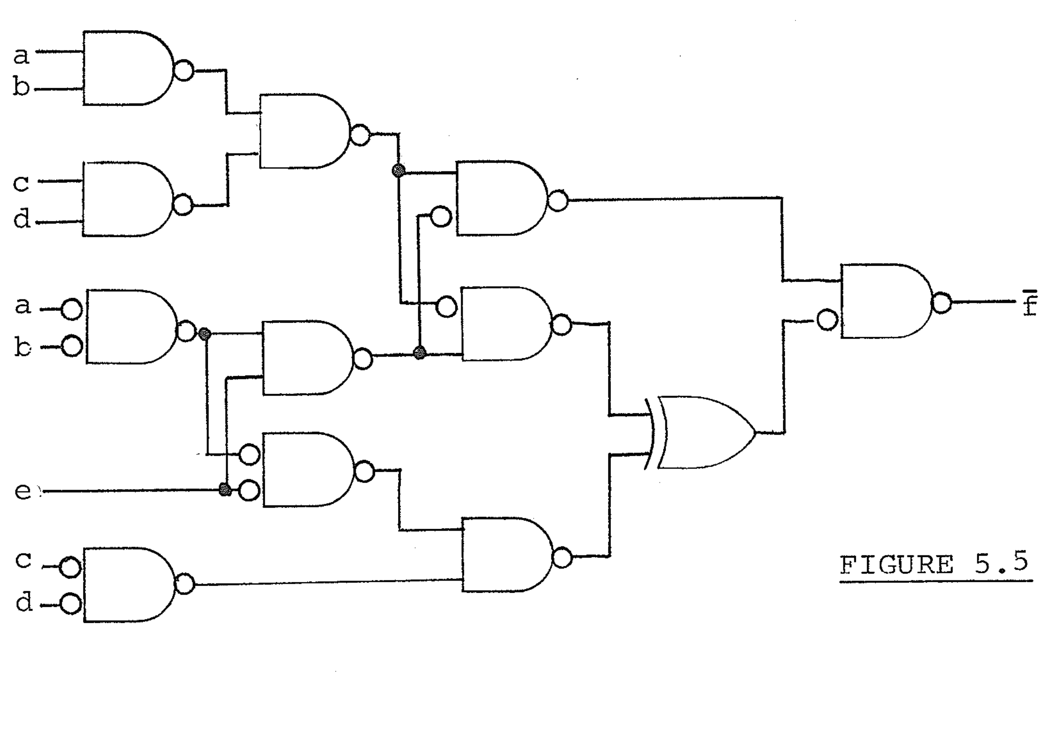


FIGURE 5.5

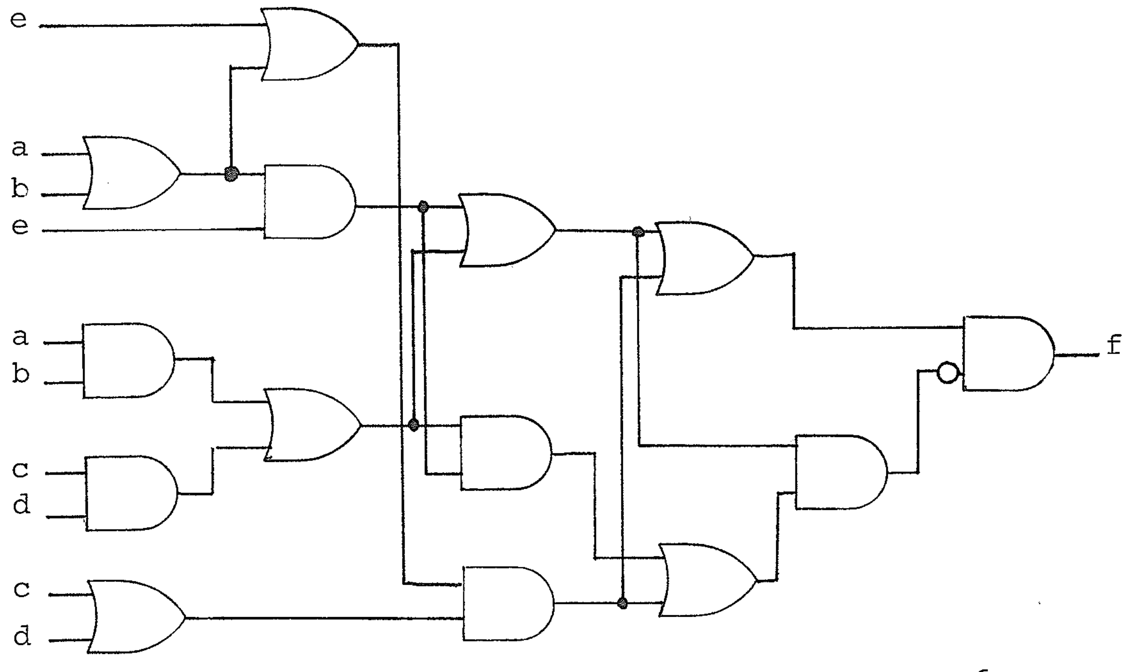


FIGURE 5.6

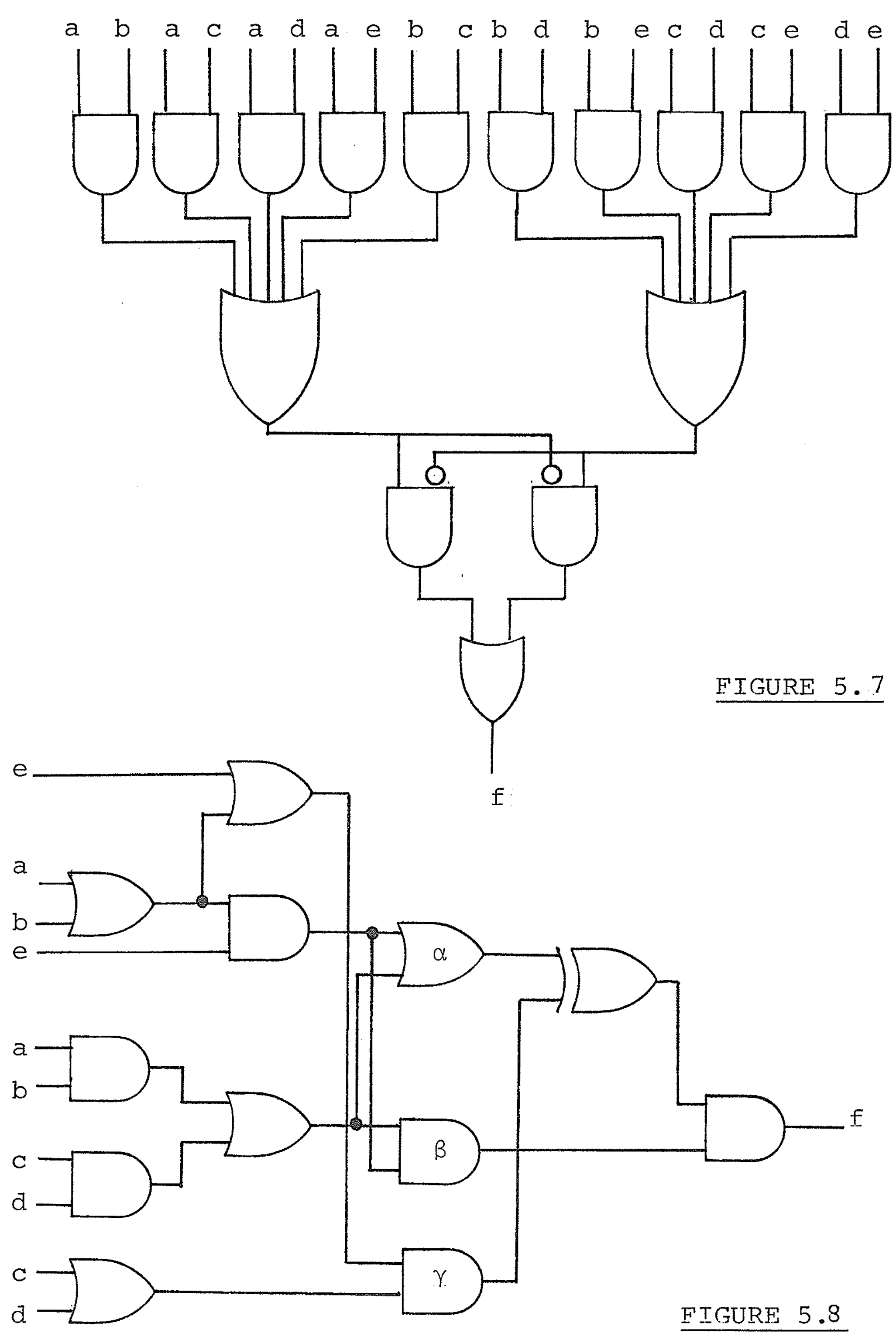


FIGURE 5.7

FIGURE 5.8

method [3] which is restricted to simple decompositions would not produce a circuit for this problem. Complex disjunctive decompositions have been found to appear quite frequently even though they are only chosen when no simple decomposition is possible. The program result is considerably simpler than the circuit used on the IBM 7090.

EXAMPLE 2.

The second example is the 'M and M' problem presented by Karp *et al* [60]. No reason for the peculiar name was given. Table 5.1 is a cube definition of the function to be realized. Barnard and Holman's [3] circuit was given in figure 5.3. To facilitate comparison this circuit is also shown as figure 5.9. Figure 5.10 is the circuit produced by the program. This circuit was obtained by determining a NAND solution for the dual problem to table 5.1. This circuit was found in 1.31 seconds on the IBM 370/158. Barnard and Holman's result was found in 68 seconds on a KDF9. The program result uses one more gate, one more level and has a cost of 21 as opposed to 20. The efficiency of our algorithm certainly outweighs the slight increase in the complexity of the circuit. The circuit presented by Karp *et al* was identical to our result except the gates labeled α and β were combined into a single three-input gate.

TABLE 5.1

a	b	c	d	e	f	g	h	γ
1	1	x	x	x	x	x	x	1
1	x	x	1	x	x	1	1	1
1	x	x	0	1	1	x	x	1
1	x	x	x	1	x	1	1	1
x	x	1	1	x	x	1	1	1
x	0	1	0	1	1	x	x	1
x	0	1	x	1	x	1	1	1
0	1	x	x	x	x	x	0	0
0	x	x	x	x	0	0	x	0
0	1	x	x	x	x	0	x	0
0	x	x	x	x	0	x	0	0
0	1	x	0	x	x	x	x	0
x	0	x	1	x	x	0	x	0
0	x	0	x	x	x	x	x	0
x	0	x	1	x	x	x	0	0
0	x	x	0	0	x	x	x	0
x	0	x	0	0	x	x	x	0
0	x	x	1	x	x	0	x	0
x	0	x	x	0	x	0	x	0
0	x	x	1	x	x	x	0	0
x	0	x	x	0	x	x	0	0
0	x	x	x	0	x	0	x	0
x	0	x	x	x	0	0	x	0
0	x	x	x	0	x	x	0	0
x	0	x	x	x	0	x	0	0

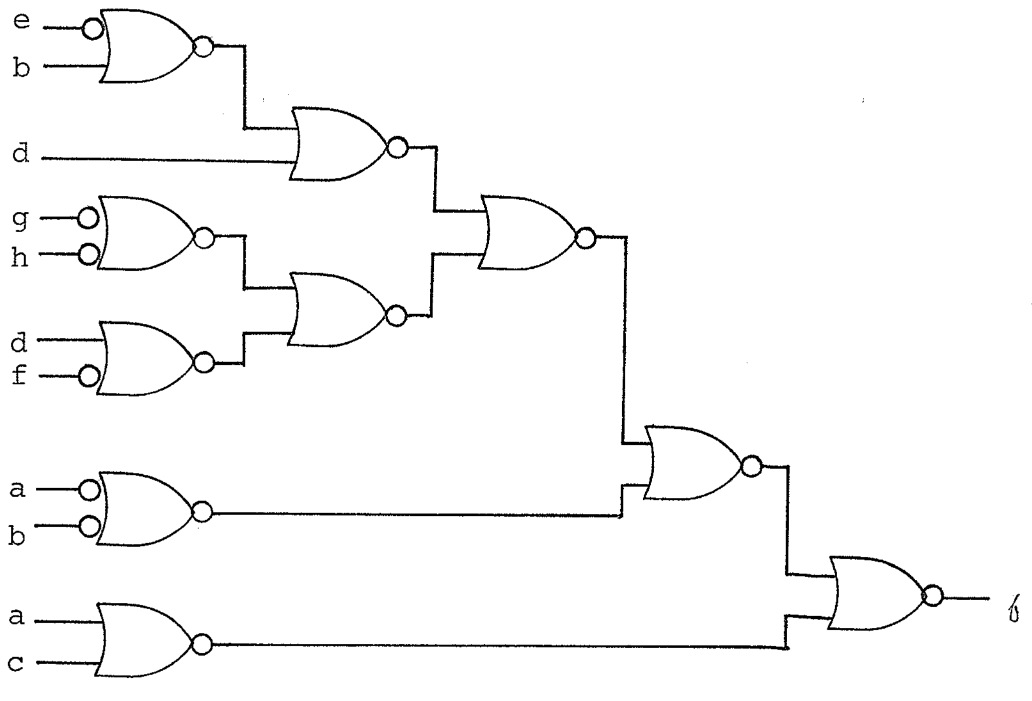


FIGURE 5.9

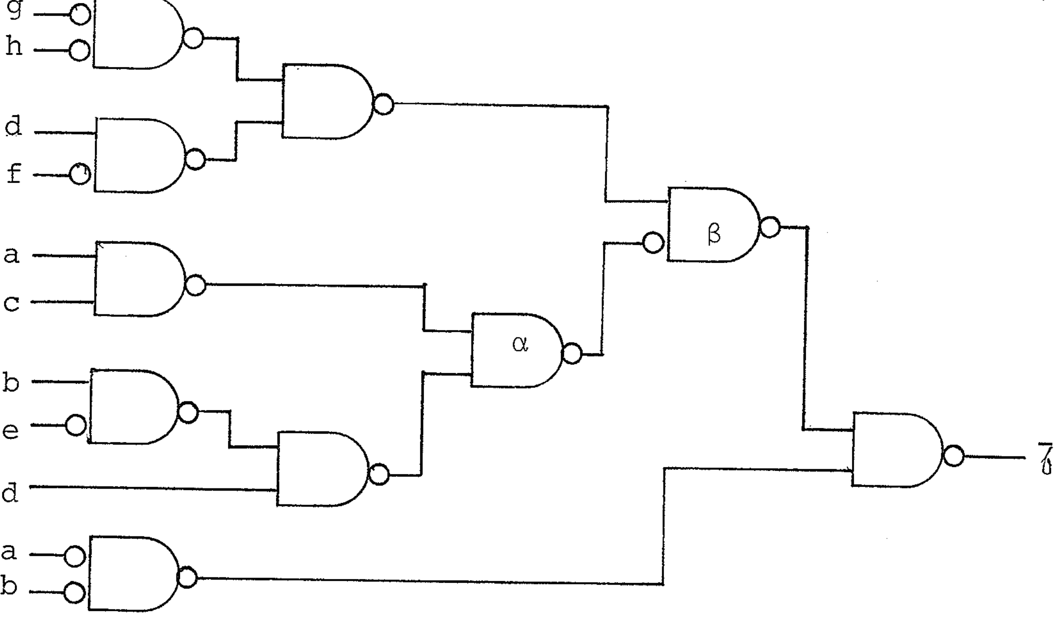


FIGURE 5.10

EXAMPLE 3.

Two-level minimization techniques have received much attention in the literature and are the most familiar methods. The circuit in figure 5.11,which was produced by the program in 1.00 second, is a realization of the function with a minimal disjunctive normal form of

$$f(a,b,c,d,e) = ab\bar{d}\bar{e} + a\bar{b}c\bar{e} + a\bar{c}d\bar{e} + a\bar{c}\bar{d}e + \bar{b}c\bar{d}e + \bar{b}\bar{c}de + \bar{a}\bar{b}cd + \bar{a}bc\bar{e} + \bar{a}b\bar{d}e + \bar{a}b\bar{c}d$$

The program result has 20 gates, nine levels and a cost of 34. The two-level realization would have 11 gates, 10 four-input and one ten-input, and a cost of 50. The program result is superior. Its only fault is the number of levels. In many cases this is not critical.

EXAMPLE 4.

A two-level realization can often be transformed into a more practical multi-level circuit. Dietmeyer and SU [12] have presented an algorithm which takes this approach. The two-level circuit is first drawn using NAND gates. Factor operations of the form shown below

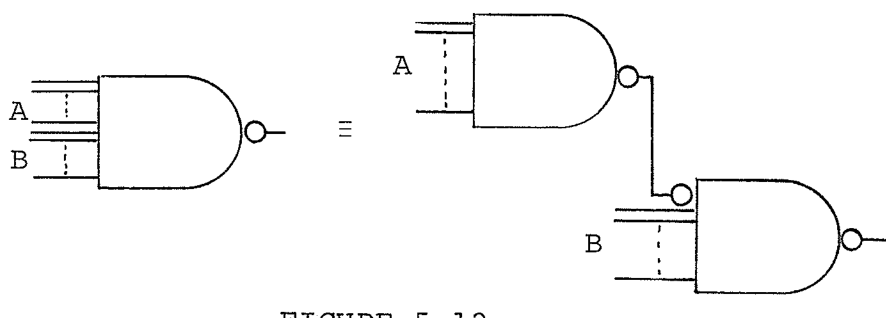


FIGURE 5.12

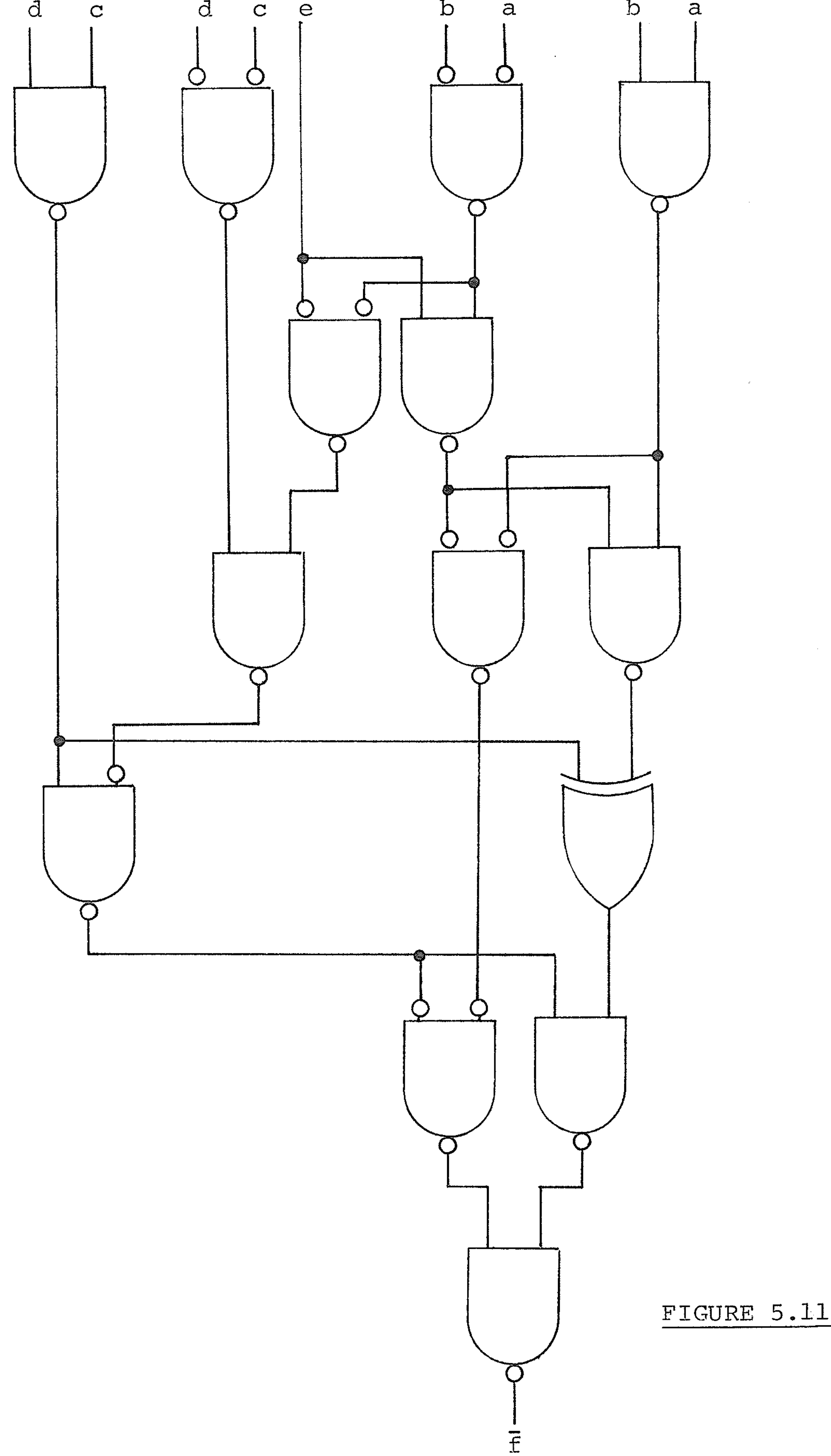


FIGURE 5.11

are applied to reduce the fan-in requirements. Care is taken to choose factors which can be employed in several operations in order to reduce the number of gates in the result. Fan-out problems are solved by cascades of NAND gates. A realization of $f(a,b,c,d) = ac + bc + \bar{c}d + \bar{a}\bar{b}\bar{d}$ found by these techniques is shown in figure 5.13. This circuit was produced by hand by the author.

The result found by our program is shown in figure 5.14. This circuit was found in 0.46 seconds. The program result uses two fewer gates, two fewer levels and has a cost of 15 as opposed to 19. The principal advantage of decomposition to factoring is the function is not required in minimal two-level form which can be extremely costly to determine. The decomposition algorithm can employ subfunctions in both true and inverted form e.g. the gate labeled α in figure 5.14. The factoring technique only uses factors in their true form.

9. MANY-VALUED EXAMPLES

The synthesis of many-valued switching circuits is a relatively new area of research. It is therefore not surprising that a search of the literature did not turn up any examples with which to evaluate our algorithm. The examples below are included to demonstrate the algorithm and also as a basis for comparing the present version of the algorithm with future variations or alterna-

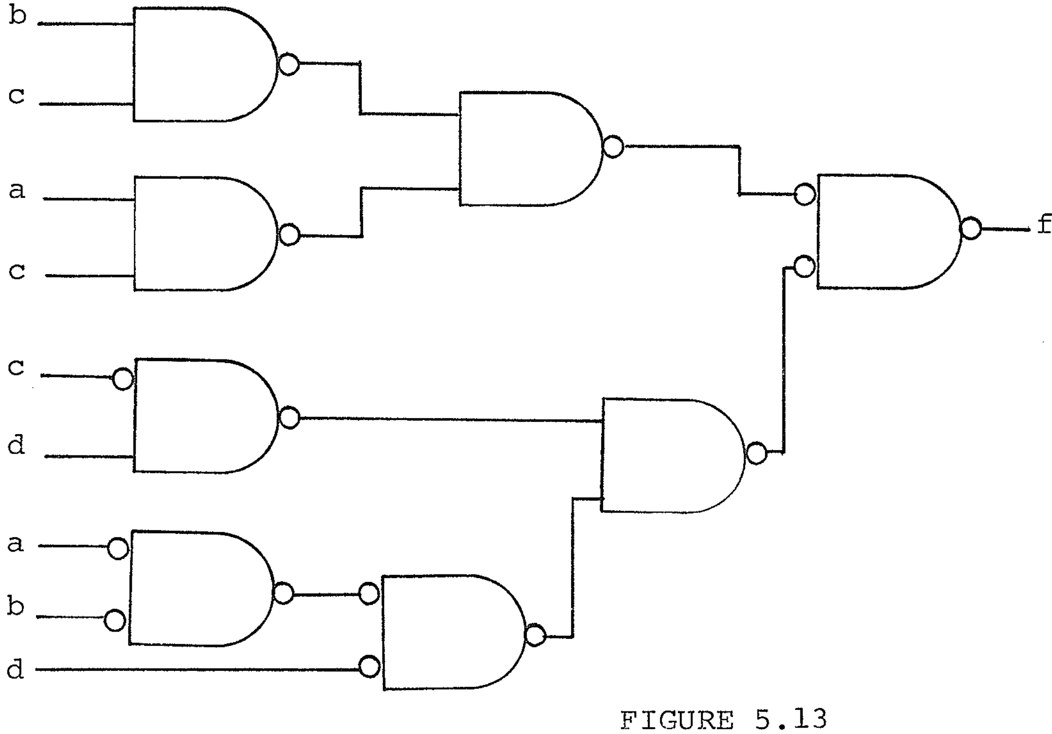


FIGURE 5.13

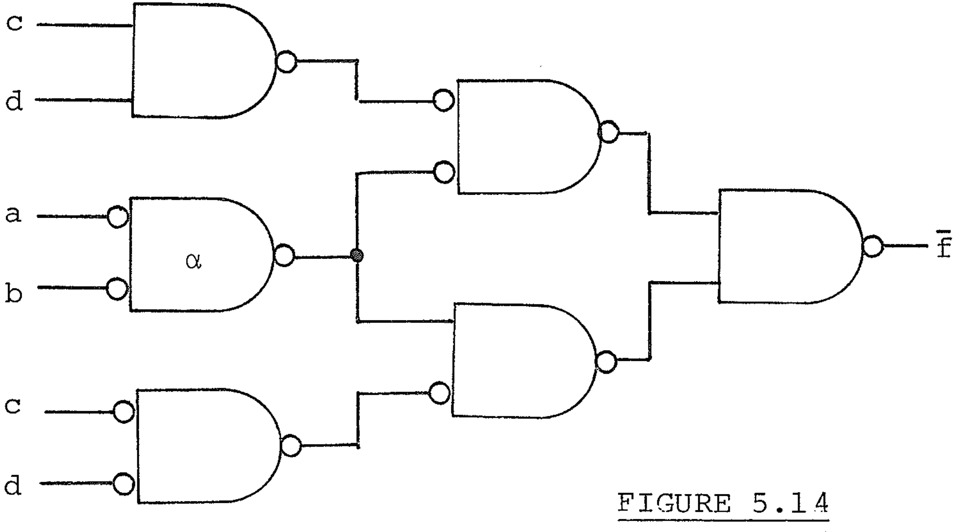


FIGURE 5.14

tive approaches which might appear.

EXAMPLE 5.

The first many-valued example is the function defined by the following table

a_1	a_2	a_3	a_4	f
0	012	012	0	0
1	012	012	0	1
2	012	012	0	2
012	0	012	1	0
012	1	012	1	1
012	2	012	1	2
012	012	0	2	0
012	012	1	2	1
012	012	2	2	2

This is the generalized condition disjunction introduced in chapter3. This function acts as a three position switch. The output is equal to a_1, a_2 or a_3 as a_4 is 0, 1 or 2 respectively. The circuit produced by our program was found in 3.00 seconds.

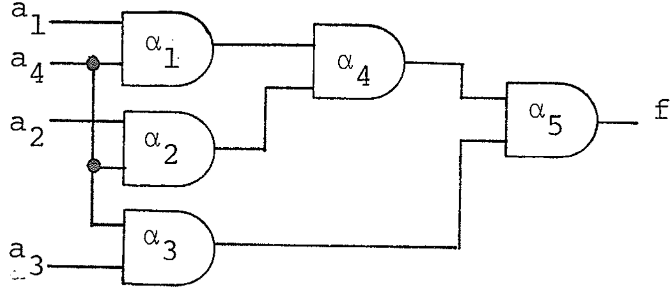


FIGURE 5.15

			a_1	
α_1		0	1	2
	0	0	1	2
a_4	1	0	0	0
	2	0	0	0

			a_2	
α_2		0	1	2
	0	0	0	0
a_4	1	2	0	1
	2	0	0	0

			a_3	
α_3		0	1	2
	0	1	1	1
a_4	1	2	2	2
	2	1	2	0

			α_2	
α_4		0	1	2
	0	0	1	2
α_1	1	1	0	0
	2	2	0	0

			α_3	
α_5		0	1	2
	0	2	0	1
α_4	1	-	1	2
	2	-	2	0

FIGURE 5.15 (cont.)

EXAMPLE 6.

The second many-valued example is a four-input circuit whose output is the number of inputs which are 0. Each input assumes a value from {0,1,2,3}. The output assumes a value from {0,1,2,3,4}. The circuit below was found by our program in 2.58 seconds.

			a_2		
α_1		0	1	2	3
	0	0	1	1	1
a_1	1	1	2	2	2
	2	1	2	2	2
	3	1	2	2	2

			a_4		
α_2		0	1	2	3
	0	0	1	1	1
a_3	1	1	2	2	2
	2	1	2	2	2
	3	1	2	2	2

			α_2	
α_3		0	1	2
	0	4	3	2
α_1	1	3	2	1
	2	2	1	0

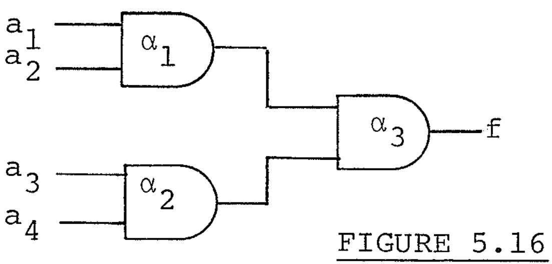


FIGURE 5.16

EXAMPLE 7.

The final example is the universal decision element defined by Muzio and the author [46]. This element is given by the table

a_1	a_2	a_3	a_4	a_5	a_6	a_7	f
0	0	0	012	012	012	012	0
0	12	2	012	012	012	012	0
0	01	1	012	012	012	012	1
0	1	01	012	012	012	012	1
0	0	2	012	012	012	012	2
0	2	01	012	012	012	012	2
1	012	012	0	0	012	012	0
1	012	012	12	2	012	012	0
1	012	012	01	1	012	012	1
1	012	012	1	01	012	012	1
1	012	012	0	2	012	012	2
1	012	012	2	01	012	012	2
2	012	012	012	012	0	0	0
2	012	012	012	012	12	2	0
2	012	012	012	012	01	1	1
2	012	012	012	012	1	01	1
2	012	012	012	012	0	2	2
2	012	012	012	012	2	01	2

The circuit found by the program in 12.41 seconds is given by

	α_1	a_3: 0	1	2
	0	0	1	2
a_2	1	1	1	0
	2	2	2	0

	α_2	a_5: 0	1	2
	0	0	1	2
a_4	1	1	1	0
	2	2	2	0

	α_3	a_7: 0	1	2
	0	0	1	2
a_6	1	1	1	0
	2	2	2	0

α_4	$a_1 = 0$	$a_1 = 1$	$a_1 = 2$
$\alpha_1 = 0$	0	0	0
$\alpha_1 = 1$	1	0	0
$\alpha_1 = 2$	2	0	0

α_5	$a_1 = 0$	$a_1 = 1$	$a_1 = 2$
$\alpha_2 = 0$	0	2	0
$\alpha_2 = 1$	0	0	0
$\alpha_2 = 2$	0	1	0

α_6	$a_1 = 0$	$a_1 = 1$	$a_1 = 2$
$\alpha_3 = 0$	1	2	1
$\alpha_3 = 1$	1	2	2
$\alpha_3 = 2$	1	2	0

α_7	$\alpha_5 = 0$	$\alpha_5 = 1$	$\alpha_5 = 2$
$\alpha_4 = 0$	0	1	2
$\alpha_4 = 1$	1	0	0
$\alpha_4 = 2$	2	0	0

α_8	$\alpha_6 = 0$	$\alpha_6 = 1$	$\alpha_6 = 2$
$\alpha_7 = 0$	2	0	1
$\alpha_7 = 1$	-	1	2
$\alpha_7 = 2$	-	2	0

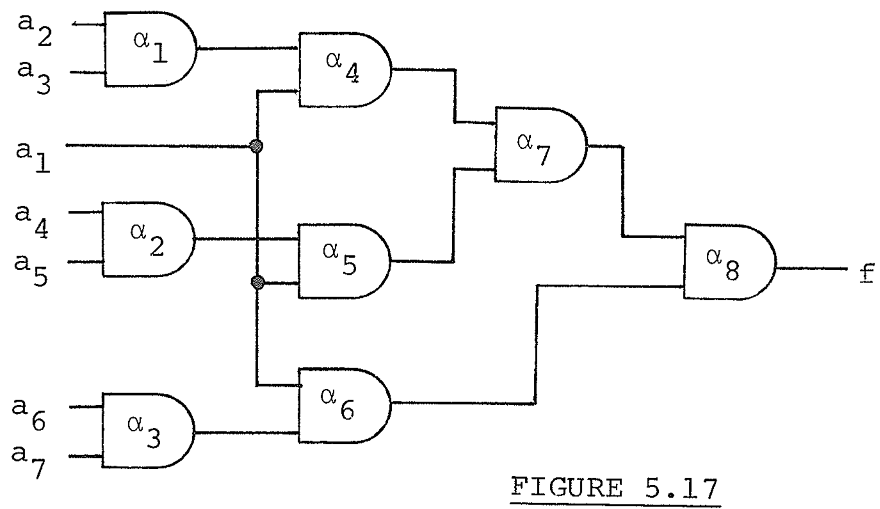


FIGURE 5.17

This is an interesting example since an implementation of each of the two-place functions comprising this circuit would form a realization of a universal decision element. This would in turn be a basis for realizing future circuits produced by our decomposition algorithm.

10. REMARKS

This chapter has demonstrated two important facts:

i) two-place decomposition is an efficient and powerful tool in circuit synthesis;

ii) good circuits can be produced by an algorithm which generates a single sequence of two-place decompositions.

Our two-valued algorithm treats a smaller class of decompositions than several of the previous algorithms and is thus theoretically more restricted. In practice it is more useful since it handles partial functions, EXOR gates and complex decompositions. This combination has not appeared in any previous implementation. The many-valued algorithm is the only multi-level many-valued synthesis procedure known to the author. Our algorithm is a preliminary attempt which will no doubt be improved with further work. The algorithm demonstrates that this approach to many-valued synthesis is a reasonable topic to pursue.

Since the algorithm is heuristic there is no way to determine a best set of selection criteria. Those presented have to date been quite successful. Only future experience, preferably in a practical design environment, will determine if better criteria exists.

One problem peculiar to the two-valued case has been identified. The circuits produced by our algorithm often have high gating levels. The major reason for this is that the algorithm only uses two-input gates. A procedure opposite to Dietmeyer and Su's factoring process

[13] can often be used to reduce the number of levels by replacing structures of the form shown in figure 5.18

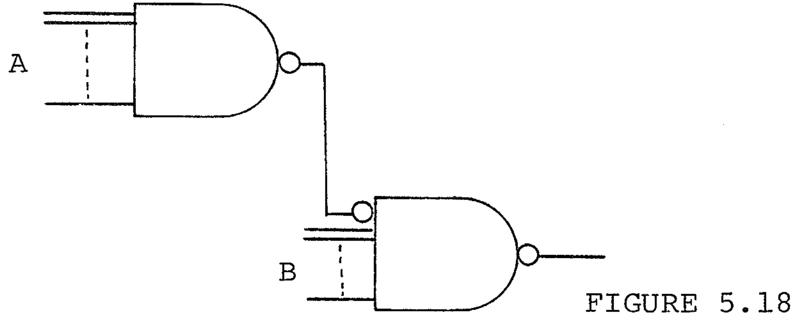


FIGURE 5.18

by a single gate (figure 5.19).

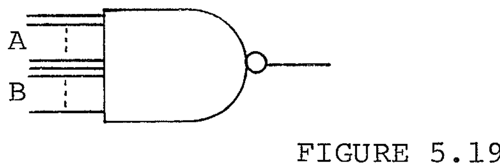


FIGURE 5.19

As an example consider figure 5.20 which is the circuit of figure 5.11. The gates have been numbered in the order they were produced by the computer program. This circuit was transformed to the circuit if figure 5.21 by making all possible replacements of multi-input gates for sequences of two-input gates. The resulting circuit uses 8 fewer gates, 3 fewer levels and has a cost of 31 as opposed to 34. The usefulness of this circuit is questionable since a seven-input gate has been introduced. Clearly, the problems of fan-in, level of gating, number of gates and cost should all be taken into account. An algorithmic procedure for this transformation process

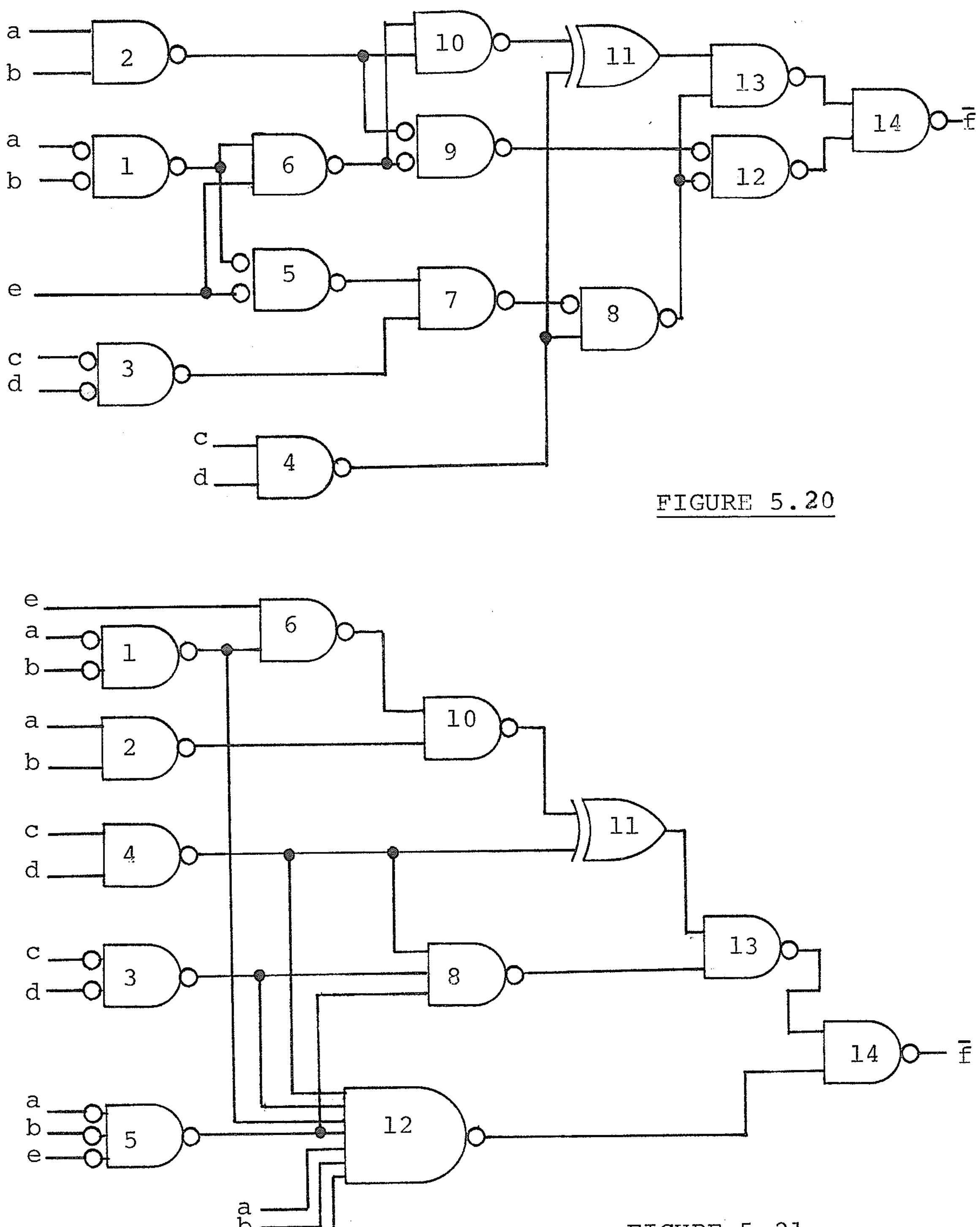


FIGURE 5.20

FIGURE 5.21

would be an excellent area for future research. There are a number of familiar gate reduction theorems for NAND circuits which should be considered in developing such a technique.

Extending the decomposition techniques to handle multi-input gates during the construction of the circuit, is not a reasonable idea. We have seen the computation involved would be a drawback. In addition, more complex selection criteria would be required to determine when the use of a multi-input gate is advisable. Decisions such as whether a single gate, figure 5.22,

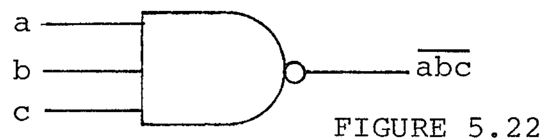


FIGURE 5.22

or the corresponding structure, figure 5.23,

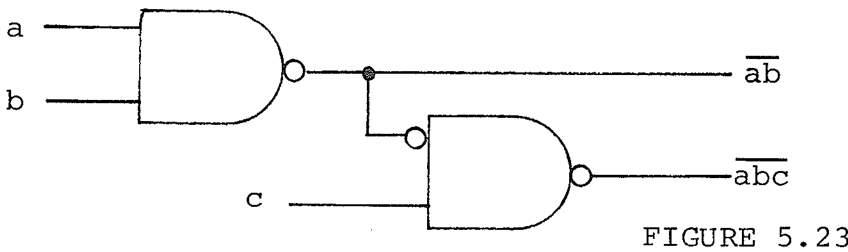


FIGURE 5.23

should be used would have to be incorporated into the algorithm. This would require some form of 'look-ahead'. In the author's opinion, it is best to consider only two-input gates in constructing the circuit. Multi-input gates should be treated as outlined above.

CHAPTER 6

Synthesis of Multiple-Output Circuits

1. INTRODUCTION

Switching circuits often realize more than one output function. For example, a four-bit parallel binary adder accepts two four-bit numbers and a carry from the previous stage and produces a four-bit result together with a carry to the next stage. Optimizing the design of such a circuit requires the output functions be considered simultaneously. Common subfunctions should be identified so that certain hardware can be employed in the realization of more than one output. This requirment makes the synthesis of multiple-output circuits much more complex than the single-output case.

A multiple-output circuit could be designed by synthesizing the output functions seperately. While the possibility of sharing gates may then be obvious, this approach does not, in general, produce an efficient realization. Better results are achieved if the identification of common subfunctions is an integral part of the synthesis procedure.

Several authors [4],[50],[72],[73],[74],[76] have considered the design of two-level two-valued switching circuits with multiple-outputs. The initial algorithm presented by Bartee [4] produces a minimal result. Each output function is realized by a circuit corresponding to a

Boolean expression in disjunctive normal form i.e. a disjunction of terms each of which is a product of variables some of which may be inverted. When it is advantageous terms are used in more than one expression. This corresponds to the hardware implementing the term being employed in the realization of more than one output. A minimal result minimizes the number of terms plus the number of variables appearing in these terms. Terms appearing in more than one expression are counted only once. These criteria minimize the number of connections in the corresponding circuit.

Bartee's algorithm requires a large amount of computation and considerable storage on the computer. Reducing these requirements has been the principal goal of the other authors who have considered this problem. Considerable savings are possible if a good rather than a minimal result is accepted.

In the single-output case it was found that a minimal two-level realization does not necessarily represent the best possible result. The number of connections can often be reduced by allowing more than two levels. The decomposition algorithm developed in chapter 5 produces a good multi-level circuit very quickly which typically has significantly fewer connections than a two-level result. Similar improvements are possible if multi-level multiple-output circuits are considered.

Su and Nam [73] have extended the single-output factoring technique due to Dietmeyer and Su [13]. This algorithm begins with a minimal or nearly minimal two-level realization and produces a multi-level NAND circuit. This algorithm is economical in terms of both computing time and storage.

Su and Nam's algorithm has the same drawbacks as the single-output algorithm. A minimal or nearly minimal two-level realization is required as input to the algorithm. The technique only handles vertex gates and can not be extended to allow the exclusive-OR or to handle many-valued functions. Factors are only employed in their true and not their inverted forms. Finally, factors are only identified if a fan-in or fan-out problem exists and factors which reduce the number of connections may be overlooked.

No multiple-output sysnthesis algorithms employing decomposition techniques have appeared. In this chapter the application of two-place decomposition to this problem is considered. An efficient multiple-output synthesis algorithm is developed for the two-valued case. Several sample solutions are presented. The results compare quite well to circuits produced by previous algorithms. The extension of this algorithm to the many-valued case is discussed.

2. MULTIPLE-OUTPUT DECOMPOSITION

In multiple-output synthesis the principal problem is the identification of subfunctions which can be used in the realization of more than one output function. A decomposition is an expression of a switching function as a composite function. In a multiple-output problem, comparing the decompositions of the output functions would seem to be a reasonable approach to identifying common subfunctions.

Karp [29] has suggested the following approach. Let $f^1(A^1)$, $f^2(A^2)$,...., $f^m(A^m)$ be the functions to be realized. At each step S a subset of $\{f^1, f^2, ..., f^m\}$ is selected. An A_λ is chosen so that $A_\lambda \subseteq A^i$ for every $f^i \in S$. The existence of such an A_λ is a criterion for choosing S.

For each $f^i \in S$ a decomposition of the form

$$f^i(A^i) = g^i(\alpha_1^i(A_\lambda), \alpha_2^i(A_\lambda), ..., \alpha_t^i(A_\lambda), A_\mu^i) \quad (1)$$

is found where $A_\lambda \cap A_\mu^i = \phi$, $A_\lambda \cup A_\mu^i = A^i$ and where each α_j^i assumes no more values than f^i. The identification of decompositions of this form was discussed in chapter 2. Such a decomposition can always be found since if no useful decomposition exists the trivial decomposition with $t = |A_\lambda|$ and each α_j^i a unique variable in A_λ is chosen.

In choosing the α_j^i it is reasonable to insist that t be as small as possible since this minimizes the number of arguments to g^i. Within this restriction it is further required that the number of distinct α_j^i be minimized by

choosing functions which may be included in more than one of the decompositions. The latter condition requires functions which are assignable for more than one f^i be determined.

In chapter 2, the assignment of the α_j^i in a decomposition of the form (1) was found to be extremely complex. No solution is known. Without this solution there is no basis for considering the identification of simultaneous assignments for a number of functions. Karp [29] has presented a partial characterization of the simultaneous assignments to two total two-valued f^i. Unfortunately, this characterization does not suggest a practical solution.

3. MULTIPLE-OUTPUT TWO-PLACE TWO-VALUED DECOMPOSITION

A nontrivial two-place decomposition of a two-valued function f(A) is an expression of the form

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

where $A_\lambda \cap A_\mu = \phi$, $A_\lambda \cup A_\mu = A$, $|A_\lambda| = 2$, α_1 and α_2 are two-valued and at least one of α_1 or α_2 is a true two-place function. A simple example will illustrate the problem of identifying a multiple-output two-place two-valued decomposition. $\beta[x_k]$ will denote a vertex function with distinquished vertex x_k.

Consider two two-valued functions

$$f^1(a,b,c) = ab + ac + bc$$

and $f^2(a,b,c) = abc + \bar{a}b\bar{c} + a\bar{b}\bar{c} + \bar{a}\bar{b}\bar{c}$.

Let $A_\lambda = \{a,b\}$ and $A_\mu^1 = A_\mu^2 = \{c\}$. The corresponding

partition matrices are

f^1 (c \ a,b)	00	01	10	11
0	0	0	0	1
1	0	1	1	1

f^2 (c \ a,b)	00	01	10	11
0	1	1	1	0
1	0	0	0	1

For both of these matrices $\nu = 4$ and f^1 and f^2 thus have nontrivial two-place decompositions with $A_\lambda = \{a,b\}$. The assignment problem is whether α_1^1, α_2^1, α_1^2, α_2^2 can be chosen so that a function is used in both decompositions.

In the above example two functions were examined for two-place decompositions relative to the same A_λ and these decompositions were then examined for a common assignment. In general, there will be m functions which may or may not have common arguments. The objective is to identify the largest subset of these functions so that each function in this subset has a decomposition relative to the same A_λ and these decompositions can be assigned common α_i.

Consider a number of two-valued functions $f^1, f^2, \ldots, f^p$ with two-place decompositions $\delta_1, \delta_2, \ldots, \delta_p$ relative to the same A_λ. These functions are identified by constructing the complete list of decompositions of the functions being considered. S a subset of $\{\delta_1, \delta_2, \ldots, \delta_p\}$ is termed a <u>mutually assignable set</u> if either

a) each $\delta_i \in S$ can be assigned to share a single true two-place function,

or b) $\delta_k \in S$ is a complex disjunctive decomposition and every $\delta_j \in S$, $j \neq k$, can be assigned to share a function with δ_k.

Mutually assignable sets identify decompositions employing common functions and hence situations where hardware can be used in the realization of more than one output function. Determining these sets is a major step in the multiple-output synthesis algorithm presented later in the chapter. A largest mutually assignable set is sought since this represents the sharing of hardware by the greatest number of functions.

The largest mutually assignable sets are constructed by an iterative procedure. Consider a number of two-valued functions $f^1, f^2, \ldots, f^p$ with two-place decompositions $\delta_1, \delta_2, \ldots, \delta_p$ each relative to the same A . Let $S = \{\{\delta_1\}, \{\delta_2\}, \ldots, \{\delta_p\}\}$. S is a set of mutually assignable sets. S' is found from S as follows

1. $S' \leftarrow \phi$
2. For each δ_i, $1 \leq i \leq p$, and $S_j \in S$, if $\delta_i \notin S_j$ determine if $\{\delta_i\} \cup S_j$ is a mutually assignable set. If it is and it does not already appear in S', add it to S'.

If the resulting $S' \neq \phi$, $S \leftarrow S'$ and determine S' as above. This process is repeated until $S' = \phi$. At this point S contains all the largest mutually assignable subsets of $\{\delta_1, \delta_2, \ldots, \delta_p\}$.

This technique requires we determine if $\{\delta_i\} \cup S_j$ is a

mutually assignable set. We could form this set and then compare the decompositions to determine if they are mutually assignable. This would require considerable computation. A more efficient approach is possible since we know S_j is mutually assignable.

There are eleven types of two-place decompositions of a two-valued function. These are listed in table 6.1. This list corresponds to the list near the end of chapter 4. Mutually assignable sets are classified according to the true two-place functions required to realize the decompositions in the set. There are eleven classes. These are listed in table 6.2.

Determining if $\{\delta_i\} \cup S_j$ is mutually assignable involves a comparison of the type of δ_i and the class of S_j. There are a number of cases:

a) δ_i is type 1 through type 5. $\{\delta_i\} \cup S_j$ is mutually assignable if, and only if, S_j is in a class containing the function required by δ_i.

b) δ_i is type 6 through type 9. $\{\delta_i\} \cup S_j$ is mutually assignable if, and only if, S_j is in a class containing one of the true two-place functions which can be assigned to δ_i.

c) δ_i is type 10 or type 11. $\{\delta_i\} \cup S_j$ is mutually assignable if, and only if, S_j is in a class containing at least one of the functions required by δ_i.

TABLE 6.1

type	$\alpha_1(a_i,a_j)$	$\alpha_2(a_i,a_j)$
1	constant	$\alpha_2=\beta[11]$
2	constant	$\alpha_2=\beta[10]$
3	constant	$\alpha_2=\beta[01]$
4	constant	$\alpha_2=\beta[00]$
5	constant	equivalence or nonequivalence
6	$\alpha_1=a_i$ or $\alpha_1=a_i'$	$\alpha_2=\beta[0-]$
7	$\alpha_1=a_i$ or $\alpha_1=a_i'$	$\alpha_2=\beta[1-]$
8	$\alpha_1=a_j$ or $\alpha_1=a_j'$	$\alpha_2=\beta[-0]$
9	$\alpha_1=a_j$ or $\alpha_1=a_j'$	$\alpha_2=\beta[-1]$
10	$\alpha_1=\beta[00]$	$\alpha_2=\beta[11]$
11	$\alpha_1=\beta[01]$	$\alpha_2=\beta[10]$

TABLE 6.2

class	assigned functions
1	$\beta[11]$
2	$\beta[10]$
3	$\beta[01]$
4	$\beta[00]$
5	equivalence or nonequivalence
6	$\beta[00]$ or $\beta[01]$
7	$\beta[10]$ or $\beta[11]$
8	$\beta[00]$ or $\beta[10]$
9	$\beta[01]$ or $\beta[11]$
10	$\beta[00]$ and $\beta[11]$
11	$\beta[01]$ and $\beta[10]$

When $\{\delta_i\} \cup S_j$ is mutually assignable its class must be determined. There are several cases:

a) δ_i is type 10 or type 11. $\{\delta_i\} \cup S_j$ is in class 10 or class 11 respectively.

b) δ_i is type 1 through type 5 and S_j is in class 1 through class 9. $\{\delta_i\} \cup S_j$ is in the same class as S_j.

c) δ_i is type 6 through type 9 and S_j is in class 1 through class 4. $\{\delta_i\} \cup S_j$ is in the same class as S_j.

d) δ_i is type 6 through type 9 and S_j is in class 6 through class 9. There are two possible results. If δ_i is type k and S_j is in class k, $\{\delta_i\} \cup S_j$ is in class k. When δ_i is type k and S_j is in class ℓ $\{\delta_i\} \cup S_j$ is in class 1 through 4. The correct class depends on which function in class ℓ can be assigned to a type k decomposition.

These results are summarized in table 6.3. A blank entry indicates $\{\delta_i\} \cup S_j$ is not mutually assignable. When the resulting set is mutually assignable the value in table 6.3 indicates its class. This table is the basis for an efficient implementation of the procedure for determining mutually assignable sets.

The synthesis procedure chooses one largest mutually assignable set according to a heuristic criterion which is described later in this chapter. Each decomposition in this set is then applied to its corresponding output func-

TABLE 6.3

type of δ_i \ class of S_j	1	2	3	4	5	6	7	8	9	10	11
1	1						1		1	10	
2		2					2	2			11
3			3			3			3		11
4				4		4		4		10	
5					5						
6			3	4		6		4	3	10	11
7	1	2					7	2	1	10	11
8		2		4		4	2	8		10	11
9	1		3			3	1		9	10	11
10	10			10		10	10	10	10	10	
11		11	11			11	11	11	11		11

tion. If the decomposition is type 6 through type 9, the two-place function chosen to realize the decomposition depends on the class of the mutually assignable set. The proper choices are listed in table 6.4.

4. MULTIPLE-OUTPUT MANY-VALUED TWO-PLACE DECOMPOSITION

The above techniques are simple, very efficient and easily programmed on the computer. Unfortunately, this approach can not be extended to the many-valued case. There are $r^{r^2} - r^r$ true r-valued two-place functions. For r = 3 this value is 19,656. It is impossible to construct tables analogous to tables 6.1 through 6.4 for such a large number of functions. Each time a decomposition is to be added to a mutually assignable set, a detailed comparison of this decomposition with the decompositions in the set is required. This comparison employs an extension of the assignment procedure for a single decomposition which was developed in chapter 3.

Consider a number of r-valued functions $f^1, f^2, \ldots, f^p$ with two-place decompositions $\delta_1, \delta_2, \ldots, \delta_p$ each relative to the same A_λ. Determining if $\{\delta_1, \delta_2, \ldots, \delta_p\}$ is a mutually assignable set is divided into two cases. First, suppose each δ_i is a simple disjunctive decomposition or a simple nondisjunctive decomposition. Each δ_i is realized with a single true two-place function. When $\{\delta_1, \delta_2, \ldots, \delta_p\}$ is a mutually assignable set this function must be the same for all δ_i.

TABLE 6.4

class of set	6	7	8	9
1		β[11]		β[11]
2		β[10]	β[10]	
3	β[01]			β[01]
4	β[00]		β[00]	
5				
6	β[00] or β[01]			
7		β[10] or β[11]		
8			β[00] or β[10]	
9				β[01] or β[11]
10	β[00]	β[11]	β[00]	β[11]
11	β[01]	β[10]	β[10]	β[01]

type of decomposition

Each δ_i has an associated incompatibility graph G_i. A function can be assigned to δ_i if, and only if, it defines a colouring of G_i. The problem is to determine a function which defines a colouring of all the G_i.

Let G denote the <u>union</u> of the G_i. G contains all the distinct vertices taken from the G_i. Two vertices are adjacent in G if, and only if, they are adjacent in some G_i.

<u>THEOREM 6.1.</u> A given function α defines a colouring of all the G_i if, and only if, α defines a colouring of G.

<u>PROOF.</u> Suppose α defines a colouring of G i.e. $\alpha[x_j] = \alpha[x_k] \Rightarrow x_j$ and x_k are not adjacent in G. By the construction of G, x_j and x_k are not adjacent in any G_i, hence, α defines a colouring of each G_i.

Suppose α does not define a colouring of G i.e. there exist adjacent vertices x_j and x_k in G such that $\alpha[x_j] = \alpha[x_k]$. By the construction of G, x_j and x_k are adjacent in at least one G_i, hence, α does not define a colouring of every G_i. <u>QED</u>.

When the δ_i are simple decompositions, G the union of the incompatibility graphs is formed and a minimal colouring of G is determined. If $\chi(G) > r$, $\{\delta_1,\delta_2,\ldots,\delta_p\}$ is a mutually assignable set where each δ_i assumes a common function. The colouring of G defines one choice for this function. If $\chi(G) > r$, $\{\delta_1,\delta_2,\ldots,\delta_p\}$ is not mutually assignable.

When $\{\delta_1,\delta_2,\ldots.,\delta_p\}$ contains a complex disjunctive decomposition a diiferent approach is taken. Each of the δ_i which are simple decompositions must be assigned one of two true two-place functions and these functions must be used to realize the complex decomposition. If there is more than one complex decomposition in $\{\delta_1,\delta_2,\ldots.,\delta_p\}$ we insist they each be assigned the same pair of functions.

Let δ_i' denote the complex decompositions and δ_i'' the simple decompositions in $\{\delta_1,\delta_2,\ldots.,\delta_p\}$. A pair of functions α_1 and α_2 can be assigned to a complex decomposition if, and only if, the function $\alpha = (\alpha_1,\alpha_2)$ defines a colouring of the incompatibility graph corresponding to this decomposition. α is a single function and theorem 6.1 applies. We thus form G' the union of the incompatibility graphs corresponding to the δ_i'. α_1 and α_2 can be assigned to each δ_i' if, and only if, $\alpha = (\alpha_1,\alpha_2)$ defines a colouring of G'.

α_1 and α_2 are not determined from G'. Each δ_i'' must be assigned either α_1 or α_2. The approach is to determine α_1 and α_2 such that each δ_i'' can be assigned one of them and to then determine if $\alpha = (\alpha_1,\alpha_2)$ defines a colouring of G'. This is accomplished by considering all possible partitions of $\{\delta_1'',\delta_2'',\ldots.,\delta_q''\}$ into two nonempty disjoint sets. For each partition we determine if the resulting sets are mutually assignable. This is accomplished as described above since the δ_i'' are simple decompositions.

When a partition resulting in two mutually assignable sets with functions α_1 and α_2 is found, we form $\alpha = (\alpha_1, \alpha_2)$ and determine if α defines a colouring of G'. If it does $\{\delta_1, \delta_2, \ldots, \delta_p\}$ is mutually assignable using the two functions α_1 and α_2.

This discussion does not include the case where $\{\delta''_1, \delta''_2, \ldots, \delta''_q\}$ is a mutually assignable set i.e. where all the simple decompositions are assigned a single function α_1. In this case, we define a graph G''. G'' has the same vertices as G'. Two vertices labeled x_i and x_j are adjacent in G'' if, and only if, they are adjacent in G' and $\alpha_1[x_i] = \alpha_1[x_j]$. We then determine if there is an α_2 which defines a colouring of G''. If such an α_2 exists $\alpha = (\alpha_1, \alpha_2)$ represents a colouring of G'. It follows that $\{\delta_1, \delta_2, \ldots, \delta_p\}$ is mutually assignable using α_1 and α_2.

5. A MULTIPLE-OUTPUT SYNTHESIS ALGORITHM

The synthesis of a multiple-output circuit is accomplished by constructing a sequence of two-place decompositions for each of the output functions. Each of these sequences represents a switching circuit realizing the corresponding output function. Mutually assignable sets of decompositions are employed in the construction of these sequences in order to use certain hardware in the realization of more than one output. The algorithm is quite straightforward. A single decomposition sequence is con-

structed for each output according to a set of heuristic decision rules. No 'back-tracking' or 'look-ahead' is employed.

The circuit to be synthesized is specified by a number of functions $f^1,f^2,\ldots,f^m$. These functions can have 'don't-care' conditions and need not have identical arguments.

The first step in the algorithm is to determine all the two-place decompositions of $f^1,f^2,\ldots,f^m$. Decompositions representing cyclic gating are removed using the techniques described in chapter 5. The remaining list of decompositions is divided into sublists of decompositions relative to the same pairs of variables. Each of these sublists is examined and for each sublist all mutually assignable sets are determined as described above.

The algorithm selects one of these largest mutually assignable sets and finds the images of the decompositions in this set. At each step in the algorithm decompositions are thus applied to some subset of the output functions. The true two-place functions required to realize these decompositions are recorded. Each image is examined for completion of the synthesis of the corrsponding output. Completed images are removed from the problem. For all remaining output functions, those to which a decomposition was applied are replaced by the images of the decompositions and the procedure is repeated from the beginning. The algorithm terminates when all the output functions

have been completely synthesized.

The algorithm is an obvious generalization of the single-output synthesis procedure. The principal difference is that a set of decompositions is selected rather than a single decomposition. This selection process and the construction of mutually assignable sets are the only new techniques. The determination of decompositions and the images of decompositions is exactly the same as in the single-output case.

The heuristic selection criteria developed in the single-output case produce very good results. They are used as the basis for the multiple-output criteria. The multiple-output criteria will be chosen so that if they are applied to a single function they yield exactly the same result as the single-output criteria. This is desirable since at some point in the synthesis of a multiple-output circuit no decompositions can be assigned common functions, and the output functions must be synthesized separately. Rather than treating this as a special case and reverting to the single-output synthesis procedure, this situation is handled directly by the multiple-output algorithm.

Simple disjunctive, simple nondisjunctive and complex disjunctive decompositions are again assigned relative costs of 1, 2 and 3 respectively. The cost of a mutually assignable set is taken to be the sum of the costs of the

decompositions in the set. This value is used to compare mutually assignable sets of the same size. The set with the lowest cost is most desirable. The cost of a decomposition is a measure of its complexity. The cost of a mutually assignable set is a measure of the total complexity of the decompositions in the set. Thus for sets containing the same nuber of decompositions the set with minimal cost intordouces the least amount of complexity into the total circuit.

As in the single-output case, the gating level of a decomposition is taken as the larger of the numbers of levels through which its two arguments have passed. The gating level of a mutually assignable set is the sum of the gating levels of the decompositions in the set. This value is used to compare mutually assignable sets with the same number of decompositions and the same cost. The set with the lowest gating level is most desirable since it introduces a minimal increase to the gating level of the multiple-output circuit.

At each step, the multiple-output assignment algorithm selects a largest mutually assignable set. This represents a sharing of hardware by the greatest number of functions. The algorithm examines all mutually assignable sets and selects one with minimal cost and within this restriction minimal gating level. If two sets have the same cost and the same gating level, the first set to be generated is chosen. The order in which the sets are generated will

be illustrated in an example later in this section.

Suppose at some stage no decompositions can be assigned a common function. In this case, the list of decompositions found is also the list of mutually assignable sets. The cost and gating level of each set are the cost and gating level of the decomposition in the set. The criteria above thus select a decomposition of minimum cost and gating level. If two decomposition both have the same cost and gating level the first one generated is selected. The chosen decomposition applies to a single function and is the decomposition which would be selected if the single-output algorithm is applied to that function.

To illustrate the operation of the synthesis algorithm we will describe a single step in the solution of a problem suggested by Hurst [27]. A complete solution to this problem will be given below in example 1. In this discussion only the generation of mutually assignable sets and the multiple-output selection criteria are considered. The other steps in the algorithm were considered in chapter 5.

The circuit to be synthesized is specified by the following table. The inputs are from a keyboard used by the British General Post Office. The keys are arranged in four rows and three columns. Depressing a key results in the arguments associated with the row and column of that key assuming the value 1. The output of the circuit is a four-bit binary number assigned to the depressed key.

a	b	c	w	x	y	z	f^1	f^2	f^3	f^4
1	0	0	1	0	0	0	0	0	0	1
0	1	0	1	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0	1	0	0
0	1	0	0	1	0	0	0	1	0	1
0	0	1	0	1	0	0	0	1	1	0
1	0	0	0	0	1	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0
0	0	1	0	0	1	0	1	0	0	1
1	0	0	0	0	0	1	1	0	1	1
0	1	0	0	0	0	1	1	0	1	0
0	0	1	0	0	0	1	1	1	0	0

The following diagram shows the layout of the keyboard.

	a	b	c
w	1	2	3
x	4	5	6
y	7	8	9
z	11	10	12

Examining f^1, f^2, f^3 and f^4 the algorithm determines a and w are redundant in every output function, x is redundant in f^1 and y is redundant in f^4. After removing these variables the decompositions in table 6.5 are found. This

TABLE 6.5

	function	arguments	type	cost
δ_1	f^1	b c	4	1
δ_2	f^2	b x	8	2
δ_3	f^3	b x	2	1
δ_4	f^4	b x	5	1
δ_5	f^1	b y	9	2
δ_6	f^2	b y	3	1
δ_7	f^3	b y	5	1
δ_8	f^1	b z	8	2
δ_9	f^2	b z	4	1
δ_{10}	f^3	b z	4	1
δ_{11}	f^4	b z	6	2
δ_{12}	f^2	c x	8	2
δ_{13}	f^3	c x	6	1
δ_{14}	f^4	c x	8	2
δ_{15}	f^1	c y	9	2
δ_{16}	f^2	c y	6	2
δ_{17}	f^3	c y	5	1
δ_{18}	f^1	c z	8	2
δ_{19}	f^2	c z	7	2
δ_{20}	f^3	c z	5	1
δ_{21}	f^4	c z	1	1

table is arranged in sublists of decompositions relative to the same pair of variables. Reading down the table, the decomposition applicable to each function are in the order described for the single-output case.

Each of these sublists is examined and for each the maximal mutually assignable set is found as described in section 3. These sets are listed in table 6.6. $\{\delta_8, \delta_9, \delta_{10}, \delta_{11}\}$ is the only set with four decompositions and is thus selected. Each decomposition is assigned the function $\beta[00]$. The choice for δ_8 and δ_{11} was determined using table 6.4. The gates required to implement these decompositions are noted and the images of the decompositions are found.

This process is repeated until the synthesis of all four outputs is completed. As in the single-output case the result of applying the algorithm is a list of gates and interconnections describing the switching circuits.

6. EXAMPLES

Since heuristic selection criteria are used, the algorithm must be evaluated empirically. A FORTRAN IV [24] program has been written and implemented on an IBM 370/158 computer using the FORTRAN H compiler [23]. This implementation is restricted to two-valued functions. The input to the program is a list of functions. The program currently accepts up to fifteen functions each with up to fifteen arguments. The total number of cubes

TABLE 6.6

set	class	cost
$\{\delta_1\}$	4	1
$\{\delta_2,\delta_3\}$	2	3
$\{\delta_4\}$	5	1
$\{\delta_5,\delta_6\}$	3	3
$\{\delta_7\}$	5	1
$\{\delta_8,\delta_9,\delta_{10},\delta_{11}\}$	4	6
$\{\delta_{12},\delta_{13},\delta_{14}\}$	4	6
$\{\delta_{15},\delta_{16}\}$	3	4
$\{\delta_{17}\}$	5	1
$\{\delta_{18},\delta_{19}\}$	2	4
$\{\delta_{18},\delta_{21}\}$	1	3
$\{\delta_{20}\}$	5	1

required to specify all the functions can not exceed 500 at any point in the algorithm. The number of functions and the number of cubes could be increased by simply changing declaration statements. Increasing the number of arguments to each function would require extensive alterations to the program.

The output of the program is a list of gates and interconnections which comprise a realization of the given functions. The gates whose outputs realize the given functions are identified. The program currently produces a circuit using NAND and EXOR gates but is easily adpated to other types of gates.

Circuits will be evaluated using the criteria employed in the single-output case i.e. number of gates, gating level and internal connection cost which will be referred to as simply the cost of the circuit. The examples are considered in detail below.

EXAMPLE 1.

The first example is the keyboard decoder described above. The circuit produced by the program is given in figure 6.1. This circuit was determined in 1.29 seconds. Hurst's solution is given in figure 6.2. This circuit was found using synthesis techniques due to Edwards [14]. It is not known to what degree a computer was used in determining this solution. Figure 6.3 shows the program result

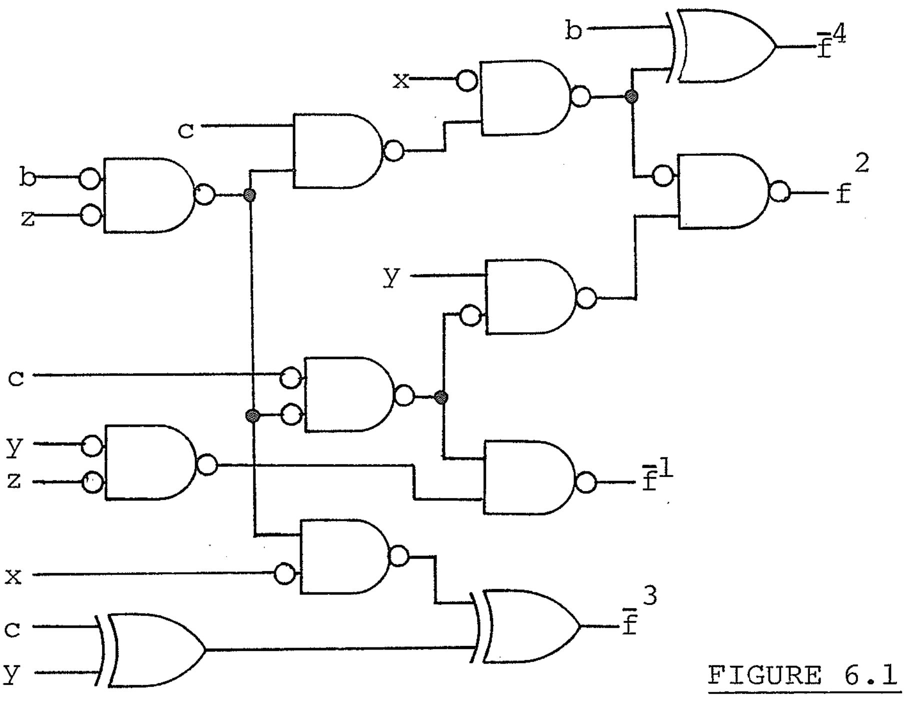


FIGURE 6.1

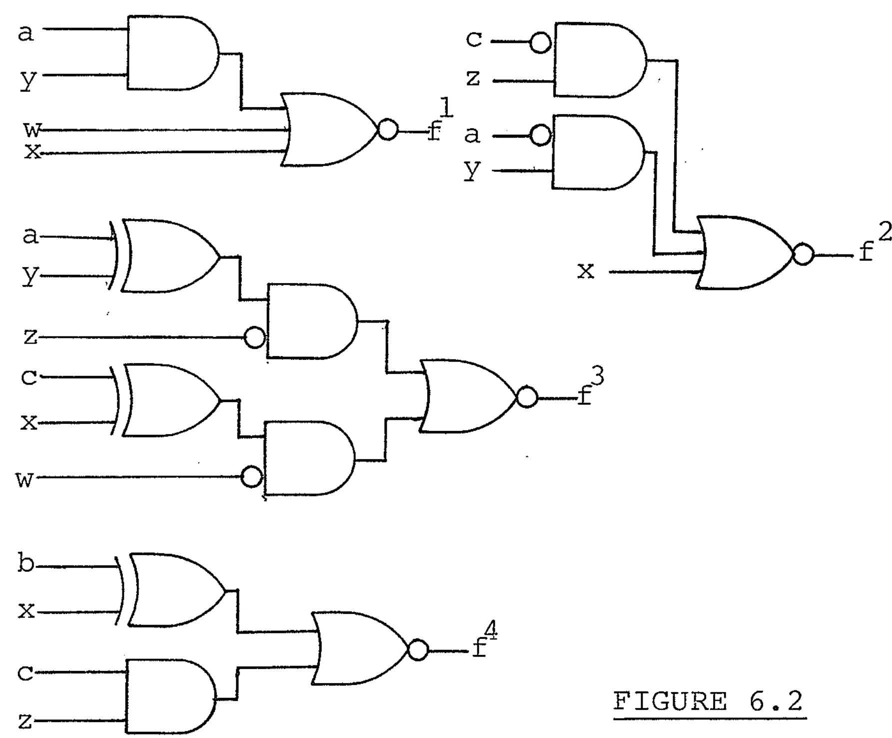


FIGURE 6.2

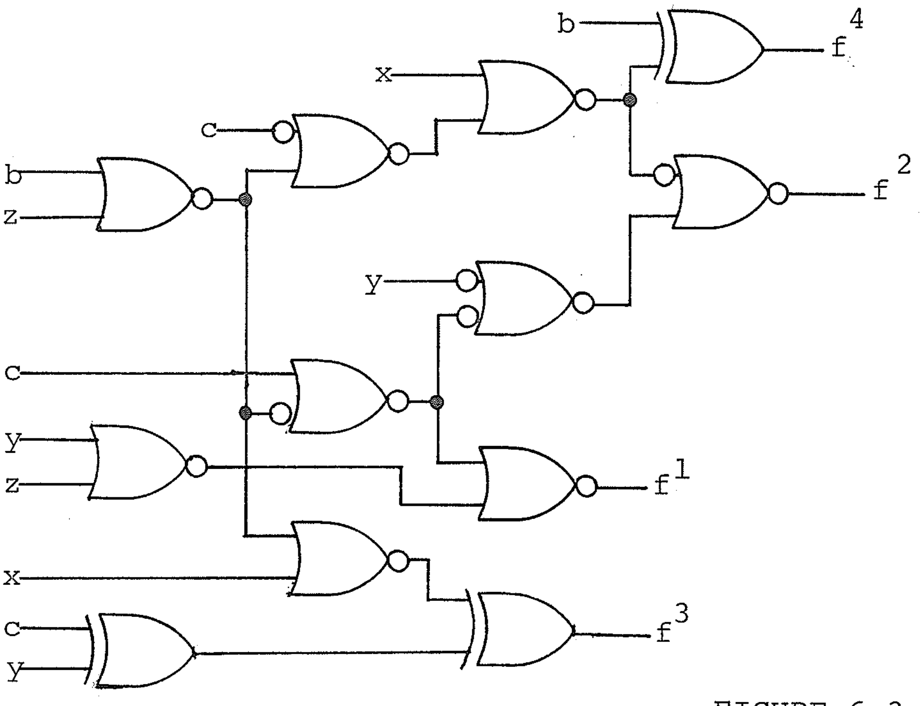


FIGURE 6.3

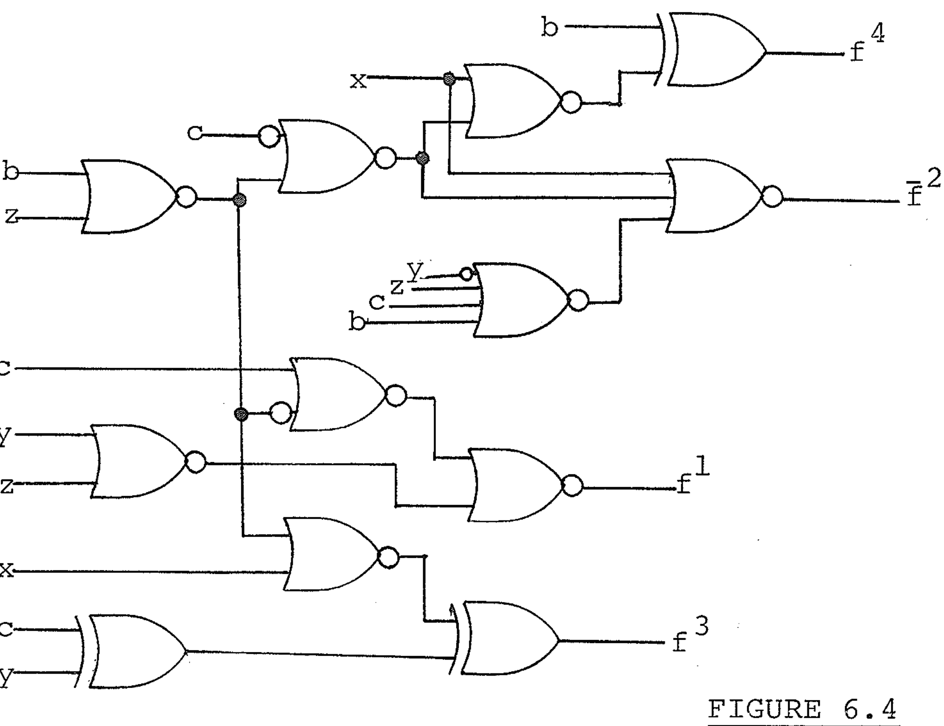


FIGURE 6.4

modified to use NOR gates. This modification was done by hand but is equivalent to altering the program to use NOR rather than NAND gates.

The principal difference between Hurst's result and our result is the number of gating levels. These values are three and six respectively. If the techniques for incorporating multiple-input gates described in chapter 5 are applied, the program result can be reduced to four levels, figure 6.4. Both Hurst's result and our result have costs of 28.

EXAMPLE 2.

The '8421' and 'excess 3' codes, table 6.7, are used to represent a decimal digit as a four-bit binary number. Figure 6.5 shows an '8421' to 'excess 3' converter produced by the program in 0.53 seconds. The inputs to the circuit a,b,c,d are a decimal digit in '8421' code with d the least significant bit. The outputs f^1,f^2,f^3,f^4 are the same decimal digit in 'excess 3' code with f^4 the least significant bit. The circuit in figure 6.6 performs the conversion from 'excess 3' to '8421'. This circuit was found in 0.64 seconds.

EXAMPLE 3.

Texas Instruments chip SN7482 [75] is a two-bit full adder. This chip accepts two two-bit binary numbers (a_2,a_1) and (b_2,b_1) and a carry-in c_0. The output is a two-bit binary result (d_2,d_1) and a carry out c_2. The

TABLE 6.7

decimal digit	'8421' code	'excess 3' code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

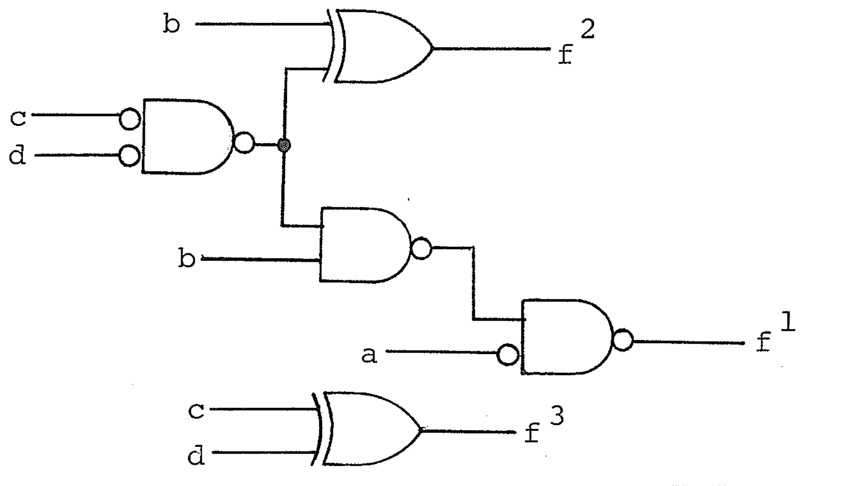


FIGURE 6.5

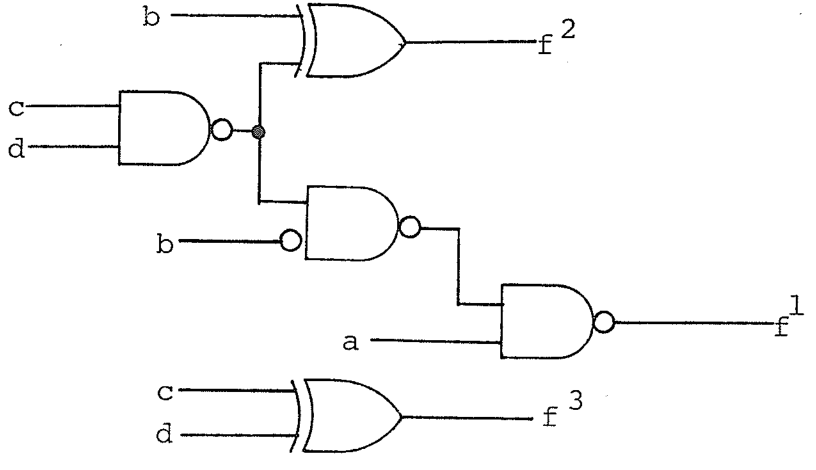
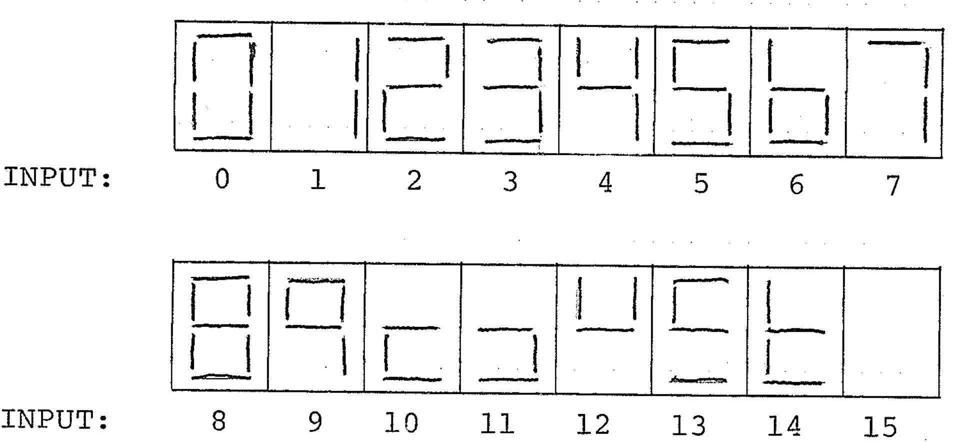


FIGURE 6.6

circuit diagram of this chip is given in figure 6.7. The circuit produced by the program is given in figure 6.8. This circuit was determined in 1.80 seconds. The program solution has five levels, one more than the Texas Instruments chip. The program solution has a cost of 24. The Texas Instruments chip has input cost 44.

EXAMPLE 4.

Several digital devices, calculators in particular, use seven segment lamps for displaying decimal digits. The decimal digit is usually stored internally in the '8421' code. A circuit is thus required to convert the '8421' code to a seven-bit code to drive the lamp. Texas Instruments chip SN7449 [75] performs this operation. It accepts a four-bit code a,b,c,d and produces a seven-bit result according to the following figures:



Note: The input is given as the decimal equivalent of the four-bit '8421' input.

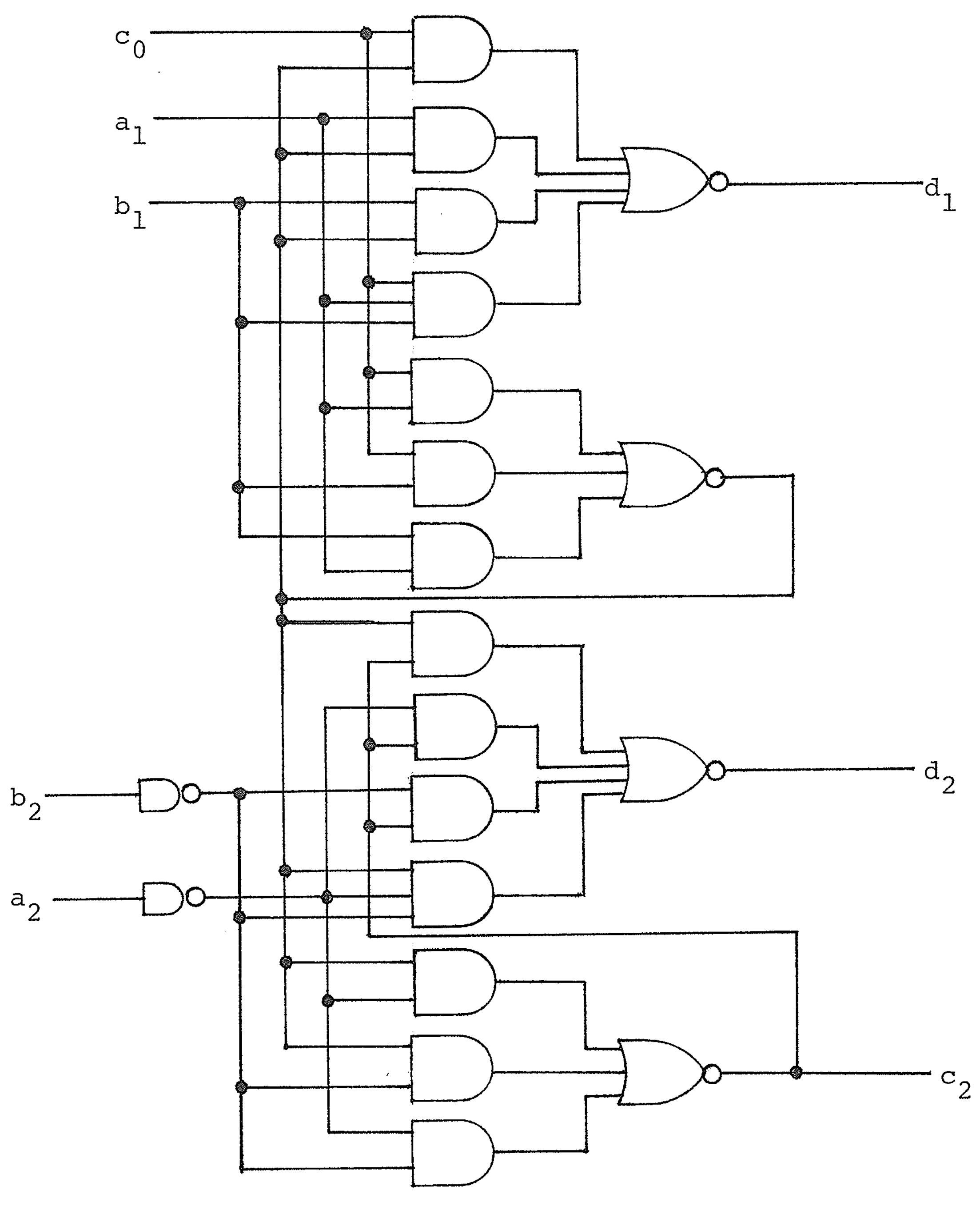


FIGURE 6.7

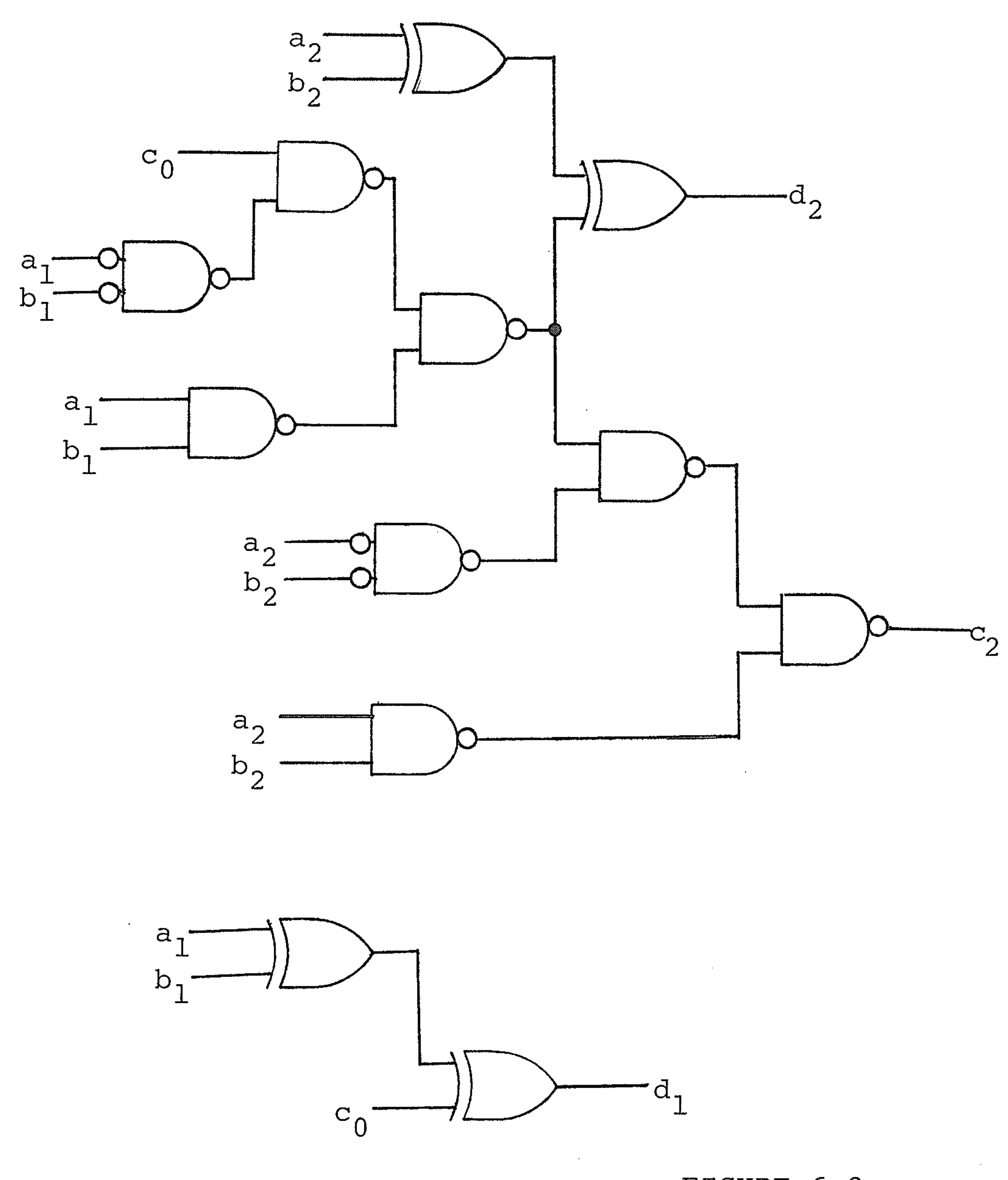


FIGURE 6.8

SEGMENT IDENTIFICATION:

```
       1
   6 |   | 2
       7
   5 |   | 3
       4
```

The chip has a fifth input z which results in a blank display when it is 0 and has no effect when it is 1. The circuit diagram of this chip is given in figure 6.9.

The circuit in figure 6.10 performs the same operation as the circuit in figure 6.9. It was determined by the program in 2.55 seconds. This circuit has a cost of 63 and eight levels. The Texas Instruments chip has a cost of 73 and three levels. In this application the number of levels is not very important since a display need only change at visual and not electronic speed.

EXAMPLE 5.

Figures 6.11 and 6.12 illustrate a binary coded decimal single digit full adder. This circuit accepts two four-bit numbers a,b,c,d and w,x,y,z which are decimal digits in '8421' code and a carry-in c_0. The output is a decimal digit f^4, f^3, f^2, f^1 in '8421' code and a carry-out c^2. c^1, g^1, g^2, g^3 and g^4 are intermediate results. The circuit was developed in two stages. The circuit in figure 6.11 accepts the two decimal digits adds them producing a decimal digit g^4, g^3, g^2, g^1 and an internal carry c^1. The circuit in figure 6.12 accepts these intermediate

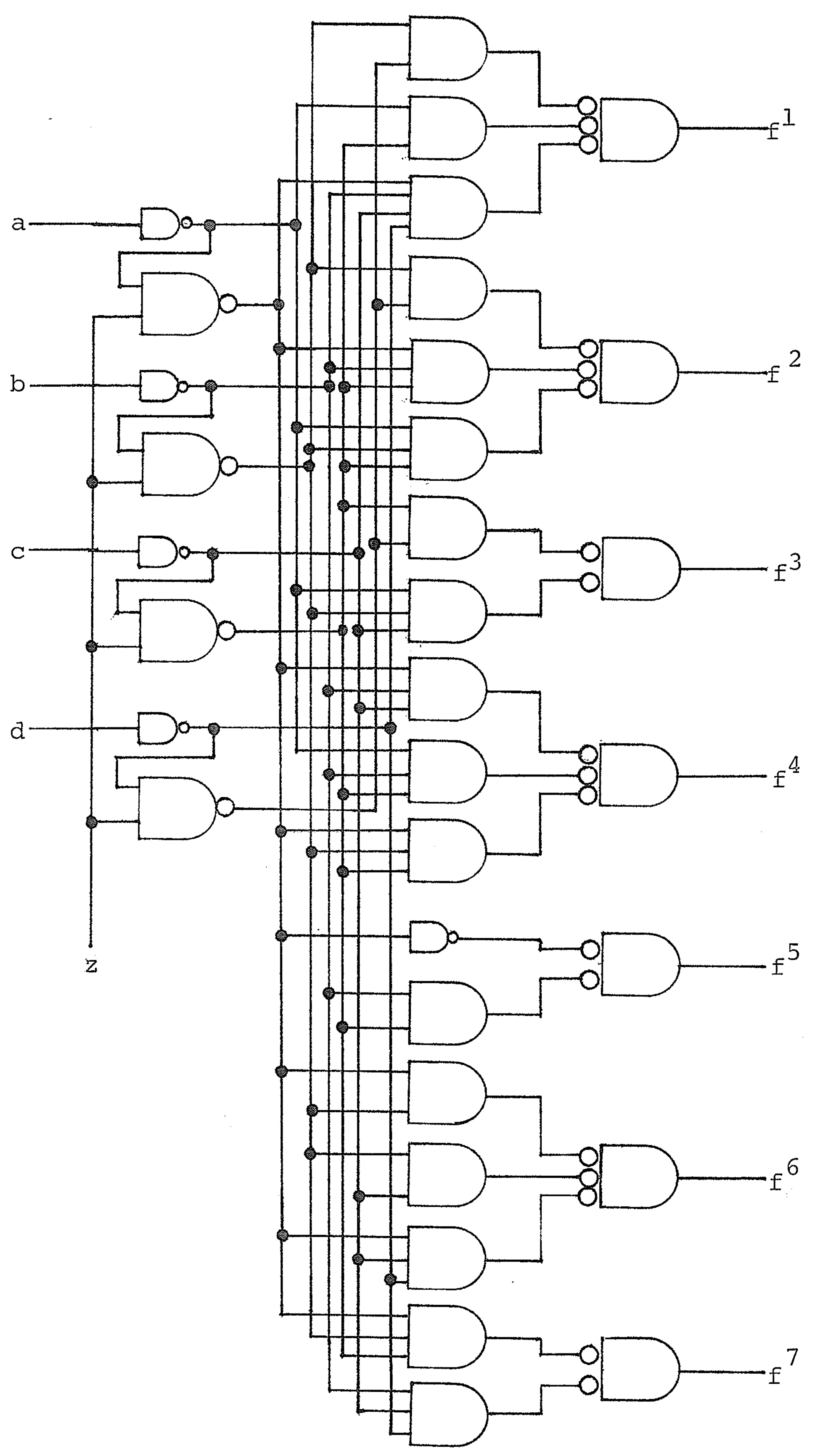


FIGURE 6.9

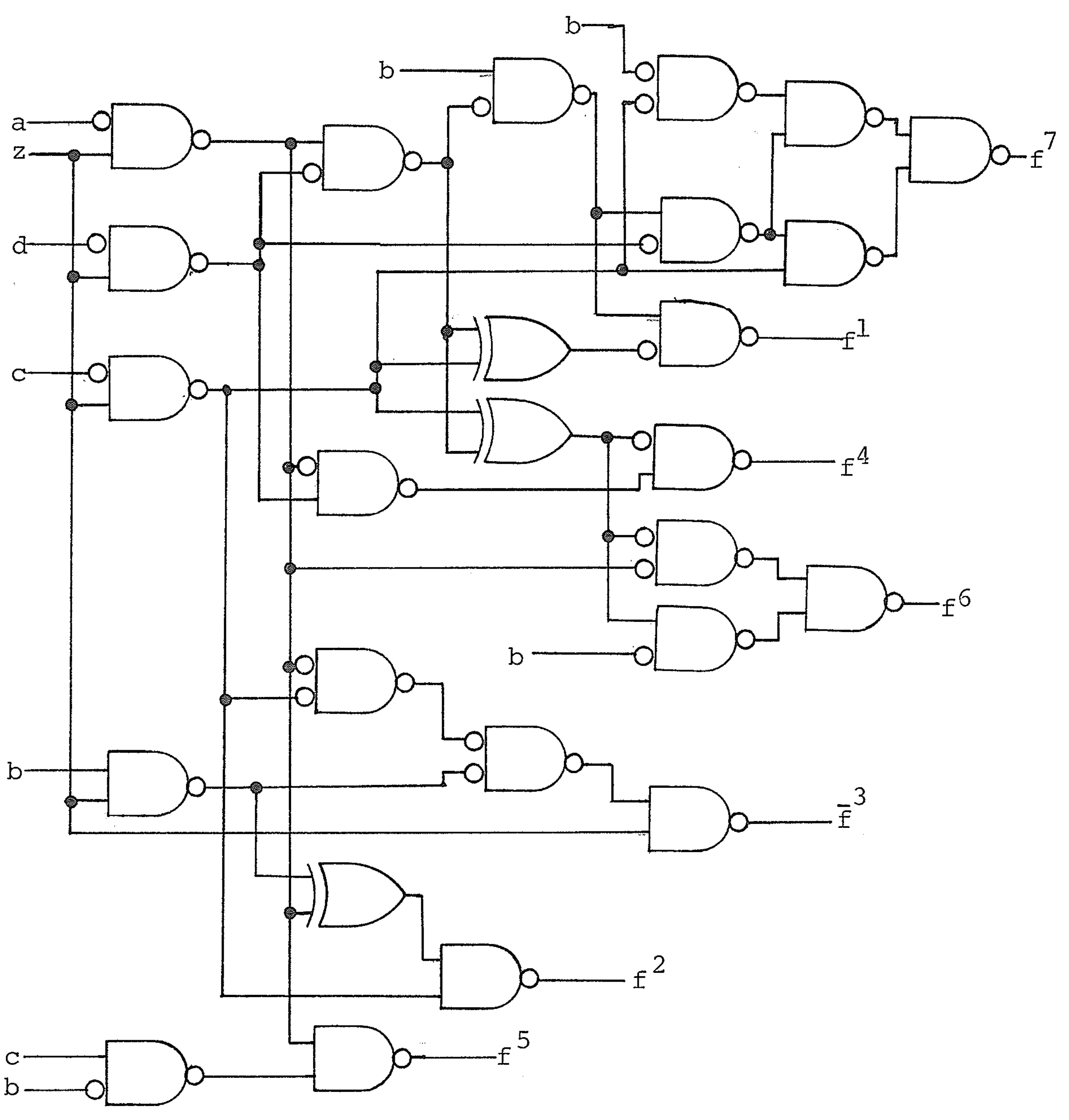


FIGURE 6.10

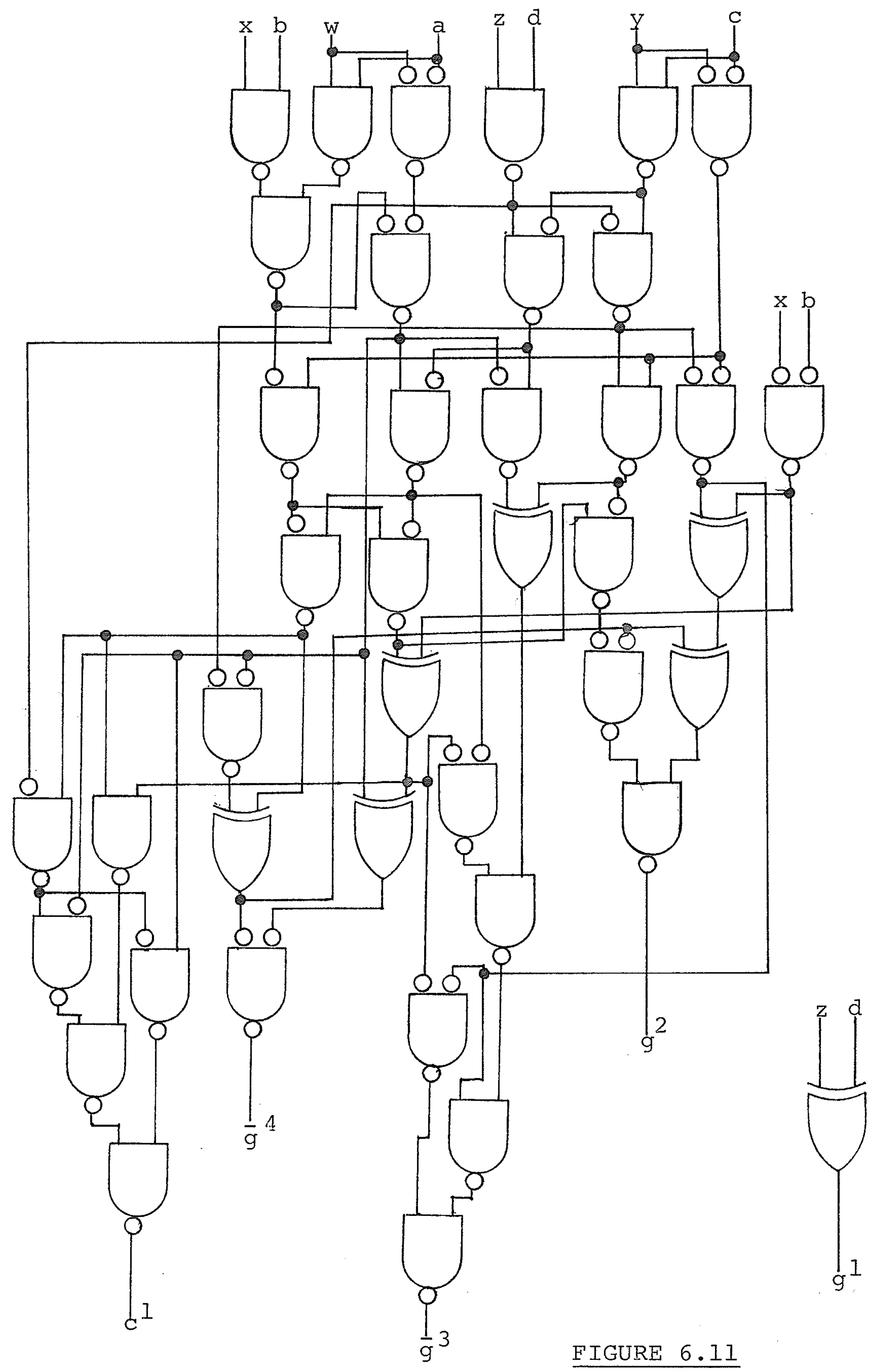


FIGURE 6.11

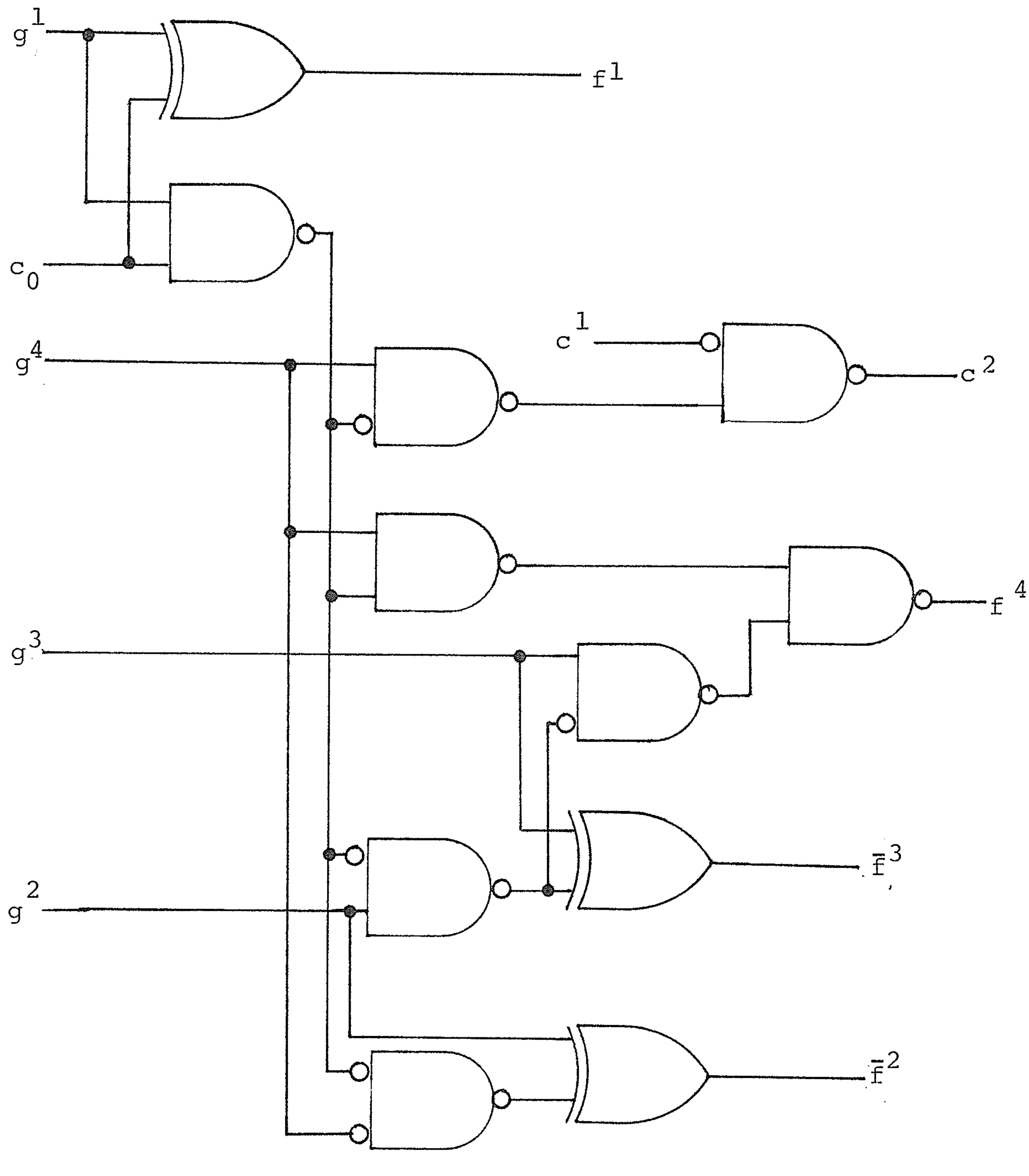


FIGURE 6.12

results and the carry-in c_0 and produces the final result f^4, f^3, f^2, f^1 and carry-out c^2. The two circuits were found in 17.15 and 1.47 seconds respectively.

Figure 6.13 shows a binary coded decimal single digit full adder [84] of the type used in calculators. This circuit operates in two stages. First, the two digits are added using a serial binary adder. If the sum is less than 10, the answer is correct. Sums of 10 through 19 must be corrected by adding 6. This is done in the second stage. Note that for suns of 10 through 15, the carry-out must also be corrected.

The circuit produced by the program has a cost of 126 and 16 levels. The calculator circuit has a cost of 69 and 21 levels. The program solution is more costly but will operate significantly faster. This speed is gained since the program treats the problem of addition as a single combinational operation rather than as a series of operations. A binary coded decimal single digit full adder is an excellent candidate for realization as a single integrated circuit chip. In this case, the additional hardware used in the program solution is insignificant while the increased speed is of great benefit. In addition the gating level could be even further reduced by introducing multiple-input gates.

d z c y b x a w

p q r c s

c_0

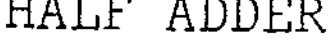
FIRST STAGE

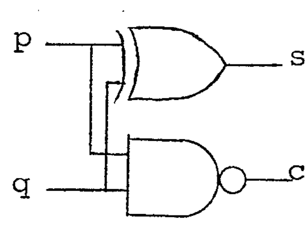
SECOND STAGE

c_2

f^1 f^2 f^3 f^4

FULL ADDER





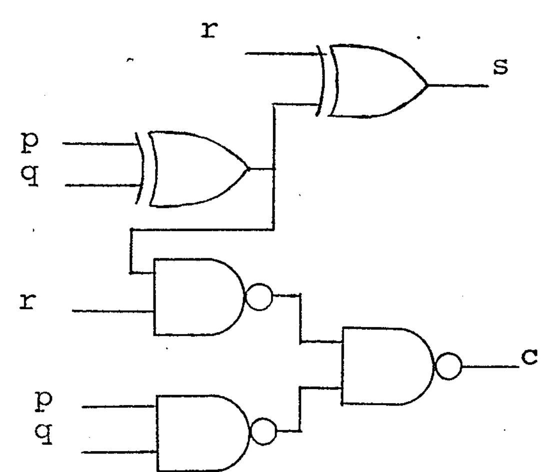


FIGURE 6.13

EXAMPLE 6.

The final example is due to SU and Nam [73] and is given by the following table:

a	b	c	d	f^1	f^2	f^3	f^4
0	0	0	x	0	1	1	0
0	0	x	0	0	1	1	0
0	x	1	0	0	1	1	0
0	0	1	1	0	1	-	0
0	1	0	0	0	0	-	0
0	1	0	1	-	0	0	0
x	1	1	1	1	1	1	1
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	1
1	1	1	0	1	0	1	1
1	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0

Note: A dash (-) indicates a 'don't-care' condition.

The program result is shown in figure 6.14. This circuit was found in 0.91 seconds. The circuit given by Su and Nam is given in figure 6.15. This result was found by applying factoring techniques. The program

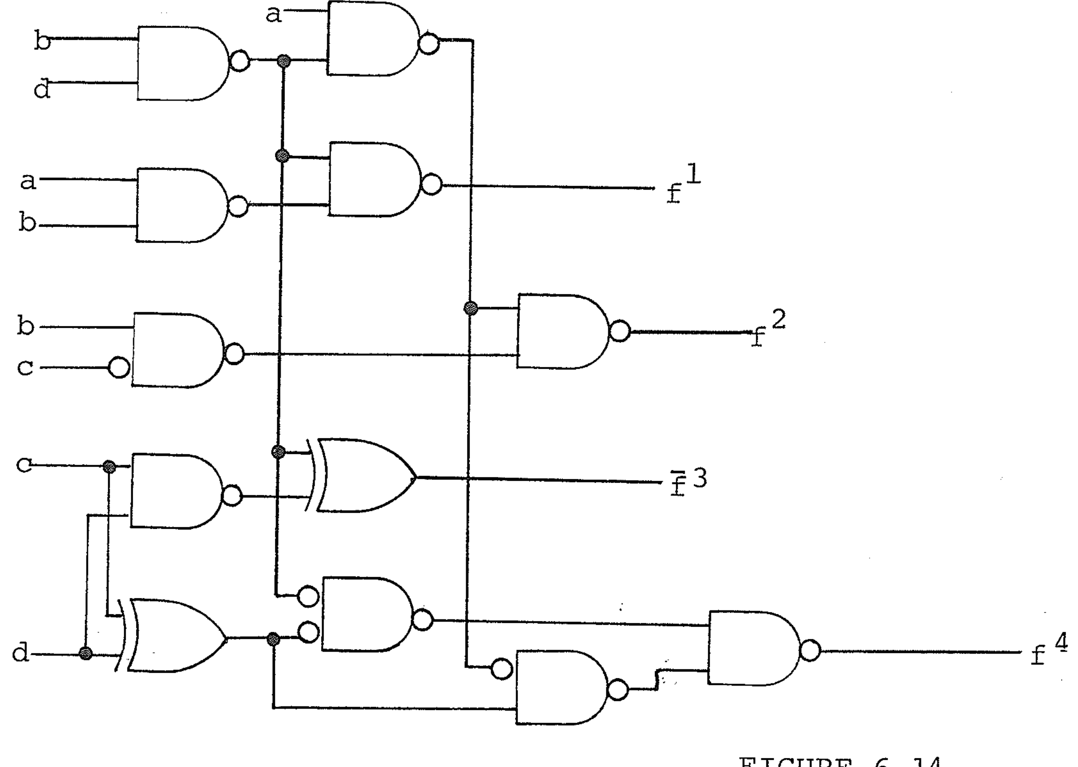


FIGURE 6.14

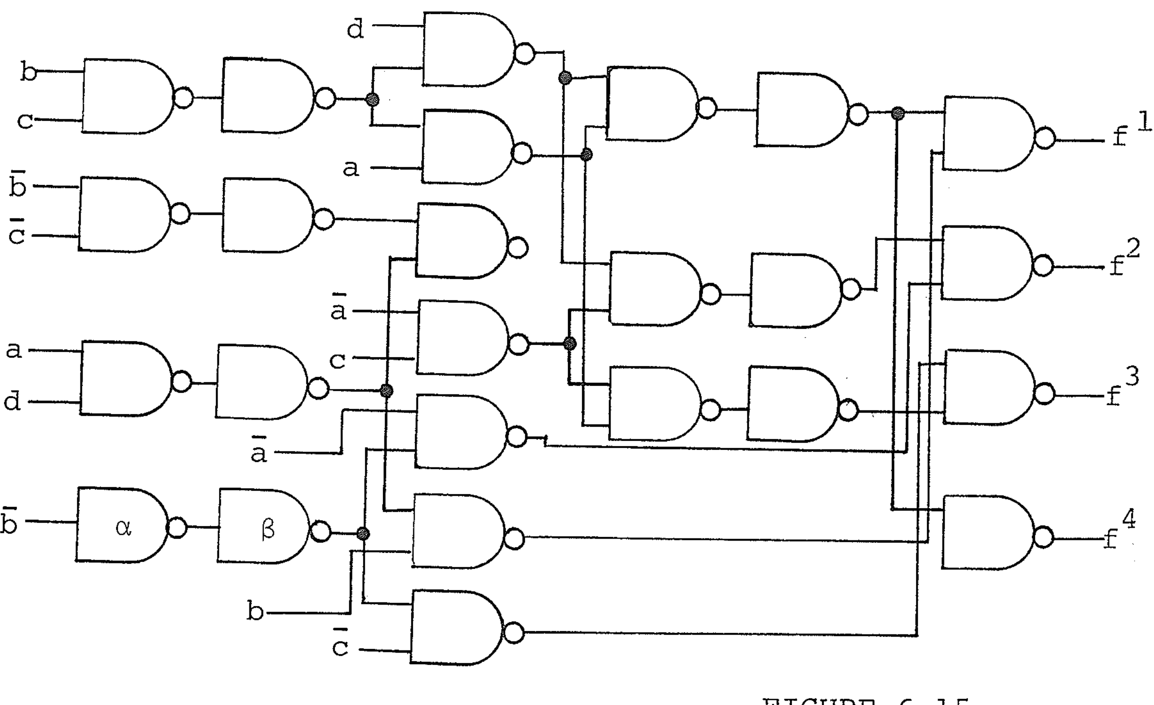


FIGURE 6.15

result has a cost of 27 and five levels. Su and Nam's result has a cost of 40 and six levels. In counting the number of connections for Su and Nam's result, the gates labeled α and β were ignored since these solve a fan-out problem which our algorithm does not consider. Including these gates would be an unfair comparison.

7. REMARKS

The above examples demonstrate that two-place decomposition is a good approach to the synthesis of two-valued multiple-output switching circuits. In particular, they show that reasonable circuits can be found by an algorithm which constructs a single sequence of decompositions with no 'back-tracking' or 'look-ahead'. They also verify that the particular heuristic criteria implemented in the program were well chosen. The simplicity of this approach results in the circuits being determined very efficiently.

The Texas Instuments '8421' to seven segment decoder, figure 6.9, is a two-level realization with certain additional initial gates to incorporate the blanking input z. The program result uses 10 fewer internal connections, a saving of 13.7%. It however uses eight levels of gating. This could be reduced using the techniques of chapter 5 but for this application is not worthwhile since the circuit is to drive a visual display.

The Texas Instruments two-bit full adder, figure 6.7, has four levels but is in fact two two-level circuits

joined together. The program result is considerably simpler. A major factor in this simplicity is that the decomposition algorithm makes effective use of EXOR gates. These are not allowed in two-level minimization techniques.

The binary coded decimal single digit full adder, figures 6.11 and 6.12, is the largest problem the program has been given. The specification for the first section, figure 6.11, was generated on the computer and consisted of five truth tables each with 100 rows. This accounts for the relatively long execution time of 17.15 seconds. This section has twelve levels. Once again this figure could be significantly reduced using the techniques of chapter 5. This example illustrates the very useful technique of solving a large problem by solving some number of smaller problems. In the next chapter, this technique is further illustrated by solving a complex problem by breaking it into several problems some of which are solved easily using the decomposition algorithm.

Figures 6.14 and 6.15 compare the decomposition program to the factoring algorithm due to Su and Nam [73] on an example presented by the latter authors. The decomposition result uses 32.5% fewer internal connections and one less level. In addition, the decomposition result has the advantage that it can accept the problem in any degree of complexity. The factoring algorithm requires the function be in minimal or nearly minimal form. This of course represents a marked difference in the required comp-

utation. Unfortunately, no exact comparison can be made since Su and Nam did not give a complete timing but only quote the time for factoring once the functions are in minimal form.

As in the single-output case, the major drawback to the circuits produced by the decomposition algorithm is the number of gating levels. This problem can usually be solved using the techniques described in chapter 5. This supports the earlier suggestion that formalizing these techniques into a computer-oriented algorithm would be a useful topic for future research.

Another interesting area for future work is the extension of the multiple-output algorithm to include many-valued functions. The identification of mutually assignable sets of two-place decompositions of many-valued functions was discussed in section 4. The techniques presented are the most obvious approach to identifying these sets. They are straightforward and would be easy to implement. Unfortunately, this approach would be extremely time-consuming. This problem should be more thoroughly investigated before an algorithm is implemented. In the single-output case, the same selection criteria were very effective for both two-valued and many-valued functions. This should also be true for the multiple-output case, since the selection criteria consider the number of true two-place functions, the number

of arguments to the image of each decomposition and the number of levels in the circuit. None of these depend on the number of values assumed by the function.

CHAPTER 7

Discussion and Conclusion

The results of this thesis are the techniques for the identification of two-place decompositions and the algorithms which employ these techniques in the synthesis of switching circuits. In this chapter we shall review these results with particular emphasis on those points which would benefit most from future research.

The design of switching circuits is a complex problem. It is unlikely a single technique will be universally applicable to all synthesis problems. A more reasonable approach is to develop a number of efficient special purpose tools which the designer can apply as each problem dictates. Two-place decomposition is one such tool. As a preliminary example of this approach, a complex design problem will be solved by manually dividing it into a number of smaller problems some of which are solved using our decomposition algorithm.

1. IDENTIFICATION OF TWO-PLACE DECOMPOSITIONS

The approaches to the identification of decompositions which have appeared in the literature were examined in chapters 2 and 3. A number of computational problems were identified. This led to the consideration of two-place decomposition. The objective was to develop techniques

which were both computationally efficient and useful in the practical design problem. It was found to be advantageous to consider two-valued and many-valued functions separately.

In the two-valued case a particularly efficient technique was developed. The major part of the required computation is performed in parallel. Each pair of cubes in the definition of the function is examined once. This is a great improvement over Roth and Karp's [59] method and a previous technique described by the author [36] where each pair of cubes must be examined for each pair of arguments. The operations required in comparing cubes are simple to implement on a computer and are very efficient. The computation which must be performed separately for each pair of variables involves the application of a number of decomposition tests each of which examines at most two cubes. The method handles partial functions with no additional effort. Except perhaps for implementation details, this method is as efficient as possible. Unfortunately, there is no generalization to two-place many-valued decompositions.

The technique for identifying two-place many-valued decompositions developed in chapter 3 requires more computation than our two-valued method. Each pair of rows in the matrix defining a function must be examined for each pair of variables. The operations involved in comparing a pair of rows are easily implemented on the computer

but are more complex than the operations used to compare two cubes in the two-valued case. They must also be performed many more times for each function considered. In describing the technique, it was suggested that the incompatible pairs of X are found and then the compatibility graph is constructed. In actual fact, these operations are combined.

The major problem in the many-valued case and certainly the one most requiring further study is the assignment of the α_i. Conditions on the number of arguments to the α_i were established in chapter 3. The problem of explicitly defining the α_i was only solved for the case where the functions are to be realized using universal decision elements. This is the simplest possible case since one-place functions need not be considered, and all two-place functions have the same hardware cost. The assignment problem was thus reduced to identifying one possible assignment and did not require examining the possible alternatives for a more desirable choice.

If the more traditional approach of realizing functions as compositions of primitive switching elements is to be used, a more complex assignment process would be required since all assignments no longer have the same hardware cost. Effective synthesis techniques are a major factor in determining the practicality of a set of primitive elements. Developing alternative assignment techniques is thus an important area for future research and may aid

in the development of practical many-valued hardware.

The many-valued decomposition techniques require the minimal colouring of certain graphs. A large number of colourings must be determined during the synthesis of a circuit and the efficiency of the graph colouring algorithm is critical. The algorithm presented has been found to be more efficient than previous algorithms which have appeared in the literature. It is beyond the scope of this discussion to give the particulars of these comparisons. A detailed analysis can be found in [37].

2. THE SYNTHESIS ALGORITHMS

The synthesis algorithms presented in chapters 5 and 6 are efficient and produce circuits which compare favourably to circuits produced by previous techniques. The principal result is the demonstration that good circuits can be produced by the generation of a single sequence of decompositions.

There are a number of interesting research topics concerning the synthesis algorithms. In chapter 5 and 6 we found using multiple-input gates to be an effective way to reduce the number of gates , the gating levels, and the number of connections in the circuits produced by our two-valued decomposition algorithm. The technique that was used introduced multiple-input gates into the completed decomposition result by replacing structures of the form shown below

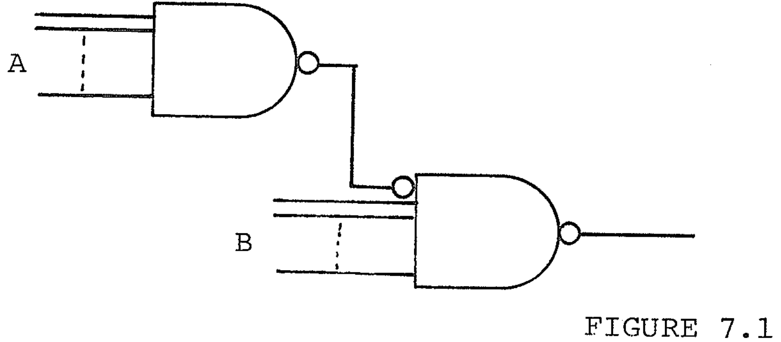


FIGURE 7.1

by a single gate.

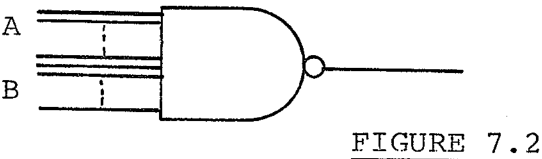


FIGURE 7.2

To date, this process has only been carried out manually. A computer algorithm combining this with similar reduction operations would be a useful design tool. This topic is currently under consideration.

In the many-valued case, the extension of the algorithm to hardware schemes other than universal decision elements and the consideration of multiple-output problems are the areas of most interest. Both these problems centre on the assignment of decompositions. The graph colouring approach presented in chapter 3 is effective for universal decision elements since any minimal colouring of the graph being considered is acceptible. Other hardware schemes may require the identification of colourings corresponding to low cost realizations in the hardware of

interest. The drawback in the multiple-output case is the amount of computation. Future research and practical experience with design problems may indicate how this can be reduced.

Sequential circuits are those whose outputs depend both on the current values of the inputs and the previous values of the outputs. They thus contain feedback loops. State table techniques for reducing a sequential design problem to a combinational problem are well known and can be found in any standard switching theory textbook. The resulting combinational problem usually involves a number of outputs. The multiple-output decomposition algorithm of chapter 6 could thus be combined with the state table techniques to form a synthesis algorithm for sequential circuits.

Although the tests performed to date indicate the selection criteria we have chosen are very effective, there is no basis for claiming they are the best possible choice. Further experiments with alternative criteria would be in order. The ideal situation would be to compare the results of applying various criteria to problems arising in a practical design environment. A theoretical characterization of the behaviour of the selection criteria would be useful and is currently under consideration.

In some cases producing a minimal result may be desirable. This will be the case when a circuit being de-

signed is to be used in a mass-produced hardware unit. The increased cost of finding a minimal result would be compensated for by the saving of hardware in each circuit produced. For such problems our heuristic selection criteria could be used to find a good initial circuit with which to begin a search similar to Roth and Karp's method [59]. Our techniques for identifying decompositions are much more efficient than Roth and Karp's techniques and would greatly reduce the cost of a complete search for a minimal solution.

3. THE EXISTENCE OF TWO-PLACE DECOMPOSITIONS

In developing the synthesis algorithms, the possibility a function may not exhibit a nontrivial two-place decomposition was ignored. Consider the function given by the tables:

a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0

a	b	c	d	f
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Applying the two-valued tests for each pair of varables we find there are no nontrivial two-place decompositions of f. Our synthesis algorithm would thus produce no circuit for this problem. Two questions arise. How often will this situation occur, and what procedure should be adopted in these cases?

Shannon [65] has shown the fraction of two-valued n-place functions with a simple disjunctive decomposition of the form

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

goes rapidly to zero as n increases. Preliminary analysis shows the same result for the fraction of r-valued functions with nontrivial two-place decompositions. This analysis has not been pursued since it is based on the assumption the functions encountered in practice will be a random selection. This is not true. The functions encountered in a practical design environment are simpler than the general run of functions for two major reasons:

i) A circuit designer has considerable freedom in the choice of functions to be realized in a given design problem, and can often choose fairly simple ones.

ii) Most operations required by digital systems are of a logically simple nature. The most important aspect of this simplicity is that a system can often be broken into a number of

small circuits. In place of a function of a large number of variables we realize many functions each of a small number of variables and then perhaps some function of these functions.

A theoretical analysis of the probability the algorithm will work is of little interest if we assume a random selection. Determining a probability the algorithm will work in a practical situation is impossible since there is no characterization of the functions that will be encountered.

An exhaustive search has established that our algorithm works for all two-valued three-place functions. There are too many two-valued four-place functions to perform an exhaustive search. It is, however, sufficient to test one function from each of the 222 equivalence classes presented by Harrison [21]. Functions in each class are equivalent up to rotation of the inputs, inversion of the inputs or inversion of the output. None of these affect the existance of a two-place decomposition. Of the 222 functions tested, nineteen exhibited no decompositions.

The failing functions are listed in table 7.1. Several attempts have been made to identify any common properties between these functions. They have been drawn in Karnaugh map form [28], as well as in the spectral

TABLE 7.1

$wxz + wy\bar{z} + \bar{w}x\bar{y}\bar{z} + \bar{w}\bar{x}yz$

$wxy + wy\bar{z} + x\bar{y}z + \bar{w}\bar{x}\bar{y}\bar{z}$

$wxz + wy\bar{z} + \bar{w}x\bar{y}\bar{z} + \bar{w}\bar{x}\bar{y}z$

$wx\bar{y} + wyz + x\bar{y}z + \bar{x}y\bar{z}$

$w\bar{x}y + \bar{w}x\bar{y} + w\bar{x}\bar{z} + xyz$

$wx + w\bar{y}z + xy\bar{z} + \bar{w}\bar{x}yz$

$wx + w\bar{y}z + xyz + \bar{w}\bar{x}y\bar{z}$

$wxz + w\bar{x}\bar{z} + wyz + xyz + \bar{w}x\bar{y}\bar{z}$

$w\bar{x}\bar{y} + wyz + xy\bar{z} + \bar{w}x\bar{y}z$

$wxy + wy\bar{z} + x\bar{y}z + \bar{w}\bar{x}yz + \bar{w}\bar{x}\bar{y}\bar{z}$

$wxz + \bar{w}\bar{x}\bar{z} + wy\bar{z} + \bar{w}\bar{y}z$

$wx + \bar{w}\bar{x}y + w\bar{y}z + \bar{w}yz$

$wx + \bar{w}\bar{x}y + w\bar{y}\bar{z} + \bar{w}yz$

$w\bar{x}y + \bar{w}xy + \bar{w}x\bar{z} + wy\bar{z} + w\bar{y}z$

$wx\bar{z} + \bar{w}xz + wyz + w\bar{y}\bar{z} + \bar{w}\bar{x}y\bar{z}$

$wx\bar{y} + wxy + wyz + \bar{x}\bar{y}\bar{z}$

$wx\bar{y} + \bar{w}xy + w\bar{x}z + xy\bar{z} + \bar{w}\bar{x}\bar{y}\bar{z}$

$wxy + \bar{w}x\bar{y} + wxz + wyz + w\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}y\bar{z}$

coefficient representation generated by a Rademacher / Walsh transform [14]. The failing cases were compared at length to cases for which the algorithm works. No insights have been gained. It is the author's oppinion that some commonality must exist. Research in this area should be pursued both to improve our decomposition algorithm and to promote a deeper understanding of the synthesis problem in general.

4. A DESIGN PROBLEM

The principal role of our decomposition algorithms will not be as self-contained design methods, but as design tools to be used by a logic designer as an aid to solving a problem. To illustrate this point we consider a complex problem which we solve by breaking it into a number of smaller problems some of which are solved using decomposition.

The problem considered is derived from the Canadian Postal Code. This code has six characters $a_1n_1a_2n_2a_3n_3$ where a_1, a_2 and a_3 are letters taken from {A,B,C,E,G,H, J,K,L,M,N,P,R,S,T,V,W,X,Y,Z} and n_1, n_2 and n_3 are decimal digits. To allow this code to be read by an optical scanner connected to the computer controlling the automatic letter sorting equipment, it is manually translated at a coding desk into a 27-bit code. This code is applied to the right-hand bottom corner of the envelope using bars of yellow phosphorescent ink to denote the positions of

the ones in the code. Numbering from left to right this code has the following format:

<u>bits</u>

bits	
0	- odd parity check bit;
1-8	- a_1n_1 coded as described below;
9-13	- a_2 coded as in table 7.2;
14-17	- n_2 coded as in table 7.2;
18-25	- a_3n_3 coded as described below;
26	- start bit which is always one and is used to align the optical scanner.

The a_1n_1 and a_3n_3 fields are coded in the same format. If the letter is L,K,M,R,J,A,H,G,S,C,B,E,Y or V the eight bits are as follows:

L - 0010****	G - 1001****
K - 0011****	S - 1010****
M - 0100****	C - 1011****
R - 0101****	B - 1100****
J - 0110****	E - 1101****
A - 0111****	Y - 1110****
H - 1000****	V - 0001****

where **** denotes the four bit representation of the digit as given in table 7.2. For example, L2 is coded as 00101001.

If the letter is T,P,W,N or Z the scheme is reversed:

TABLE 7.2

L	00010	0	1010
K	00011	1	0010
M	00100	2	1001
R	00101	3	0011
J	00110	4	1011
A	00111	5	0101
H	01000	6	0110
G	01001	7	0111
S	01010	8	1101
C	01011	9	1110
B	01100		
E	01101		
Y	01110		
V	10001		
T	10100		
P	11100		
W	11000		
X	10011		
N	10110		
Z	11010		

T - ****0100 N - ****0001

P - ****1100 Z - ****0000

W - ****1000

For example, T9 is coded as 11100100.

If the letter is X the code is taken from the following list:

X0 - 00010001 X5 - 01001100

X1 - 00010100 X6 - 11000001

X2 - 00011100 X7 - 11000100

X3 - 01000001 X8 - 11001100

X4 - 01000100 X9 - 10000100

Some examples should clarify the code.

		a_1n_1	a_2	n_2	a_3n_3	
R3N 0N8 -	1	01010011	10110	1010	11010001	1
E3B 5A3 -	0	11010011	01100	0101	01110011	1
S4T 7X6 -	1	10101011	10100	0111	11000001	1

The reason for this rather awkward coding scheme is not known. It does however present an interesting design problem. We shall design a circuit which given an eight-bit code, determines if it is a valid a_1n_1 (or a_3n_3) field and, if it is, determines the corresponding five-bit letter code and four-bit digit code as given in table 7.2. This circuit would be useful as an interface between the optical scanner and the control computer. The translation

is currently done by the software on the control computer.

Initially, we constructed an eight-argument function which is one when its arguments represent a valid code, and zero otherwise. This function has no nontrivial two-place decompositions. We next examined the structure of the code in depth.

Let $b_1, b_2, \ldots, b_8$ be the bits representing the $a_1 n_1$ field. b_1, b_2, b_3, b_4 can represent a letter, a digit or the beginning of an X code. Consider f^1, f^2 and f^3 given by

b_1	b_2	b_3	b_4	f^1	f^2	f^3
0	0	0	0	0	0	0
0	0	0	1	1	0	1
0	0	1	0	1	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	1	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	0
1	1	1	1	0	0	0

f^1 is one for the assignments to b_1, b_2, b_3, b_4 used as letter codes. f^2 is one for the assignments to b_1, b_2, b_3, b_4 used as digit codes. f^3 is one for the assignments to b_1, b_2, b_3, b_4 which appear in X codes.

Consider f^4 and f^5 given by

b_5	b_6	b_7	b_8	f^4	f^5
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0

f^4 is one for the assignments to b_5, b_6, b_7, b_8 used as numeric codes. f^5 is one for the assignments to b_5, b_6, b_7, b_8 used as letter codes. The assignments to b_5, b_6, b_7, b_8 used as X codes present a problem since they are also used as letter codes. This problem will be solved by the code translation functions described below.

If f^1 and f^4 are one, the code is valid. b_5,b_6,b_7,b_8 represent the correct digit code. b_1,b_2,b_3,b_4 must be translated to the corresponding five-bit letter code. This code is f^6,b_1,b_2,b_3,b_4 where f^6 is given by

b_1	b_2	b_3	b_4	f^6
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
0	0	0	1	1

If f^2 and f^5 are one, the code is valid. b_1,b_2,b_3,b_4 represent the correct digit code. b_5,b_6,b_7,b_8 must be translated to the corresponding five-bit letter code. This code is $1,f^7,f^8,f^9,0$ where f^7,f^8 and f^9 are given by

b_5	b_6	b_7	b_8	f^7	f^8	f^9
0	1	0	0	0	1	0
1	1	0	0	1	1	0
1	0	0	0	1	0	0
0	0	0	1	0	1	1
0	0	0	0	1	0	1

If the arguments are a valid X code, f^3 and f^5 are one but the converse does not hold. For example, 10000001 is not a valid X code. There are 20 assignments to $b_1, b_2, \ldots, b_8$ for which f^3 and f^5 are both one. Ten are valid X codes. We define the functions which determine the digit for an X code for all 20 assignments. Invalid X codes will result in a digit code of 0000 which is invalid. For an X code the digit code is $f^{10}, f^{11}, f^{12}, f^{13}$ defined as follows

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	f^{10}	f^{11}	f^{12}	f^{13}
0	0	0	1	0	0	0	1	1	0	1	0
0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	0	1	0	0	1
0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	1	1
0	1	0	0	0	1	0	0	1	0	1	1
0	1	0	0	1	1	0	0	0	1	0	1
0	1	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	1	1	0
1	1	0	0	0	1	0	0	0	1	1	1
1	1	0	0	1	1	0	0	1	1	0	1
1	1	0	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	1	1	1	0
1	0	0	0	1	1	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0

$b_1, b_2, \ldots, b_8$ represent a valid X code if $f^3 \cdot f^5 \cdot (f^{10} + f^{11} + f^{12} + f^{13}) = 1$. The functions f^1 through f^{13} were given to our multiple-output program as a single problem. The resulting circuit is given in figure 7.3. This circuit was found in 5.81 seconds.

To complete the problem a network is required which employs f^1 through f^{13} to determine if $b_1, b_2, \ldots, b_8$ represent a valid code and to choose the correct five-bit letter code and four-bit digit code. The network given in figure 7.4 was easily constructed by hand. The inverters on the g^1, g^4 and g^5 lines generate the letter code for an X code.

This example illustrates how the decomposition algorithms can be employed in a complex problem. An excelent area for future work would be to integrate our decomposition algorithm and other useful techniques into an interactive circuit design system. Such a system would make efficient use of the computer's speed and accuracy as well as the designer's experience and intelligent analysis of the problem.

5. CONCLUSION

We have shown two-place decomposition is a reasonable approach to the synthesis of combinational switching circuits. In the two-valued case, algorithms were presented for both the single and multiple-output problems. No previous multiple-output algorithm employing decomposition

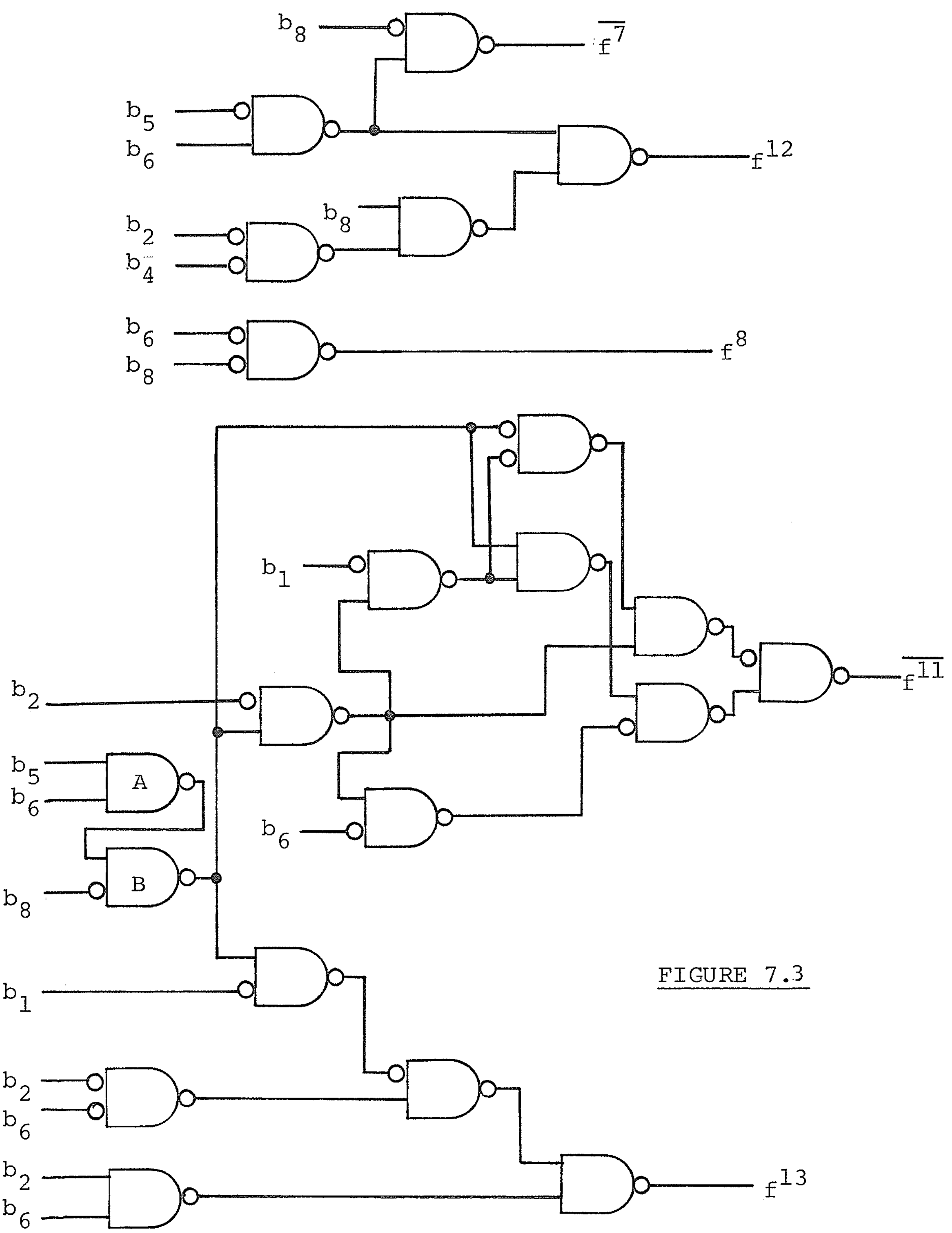
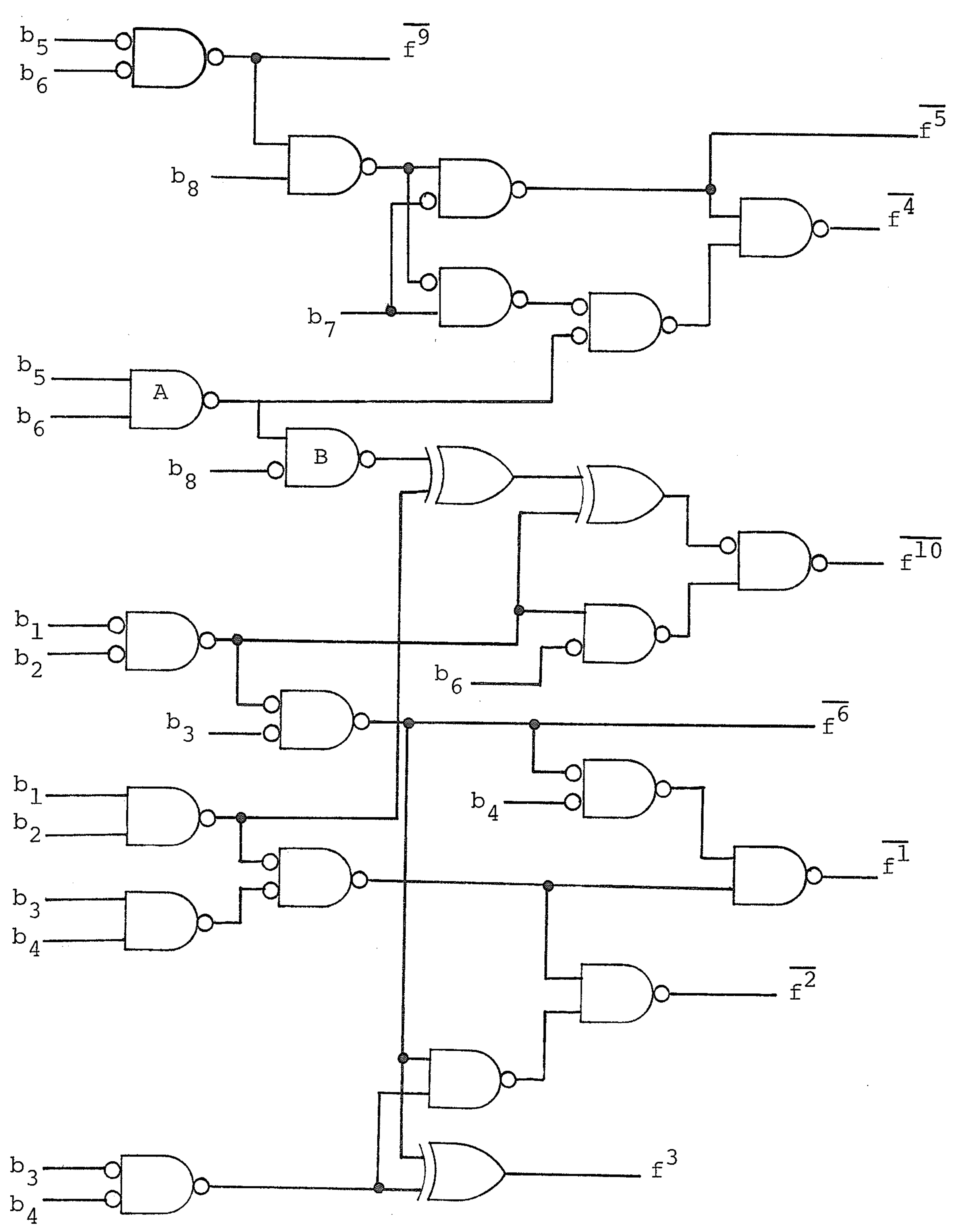


FIGURE 7.3



NOTE Gates A and B are also on the preceeding page.

FIGURE 7.3 (continued)

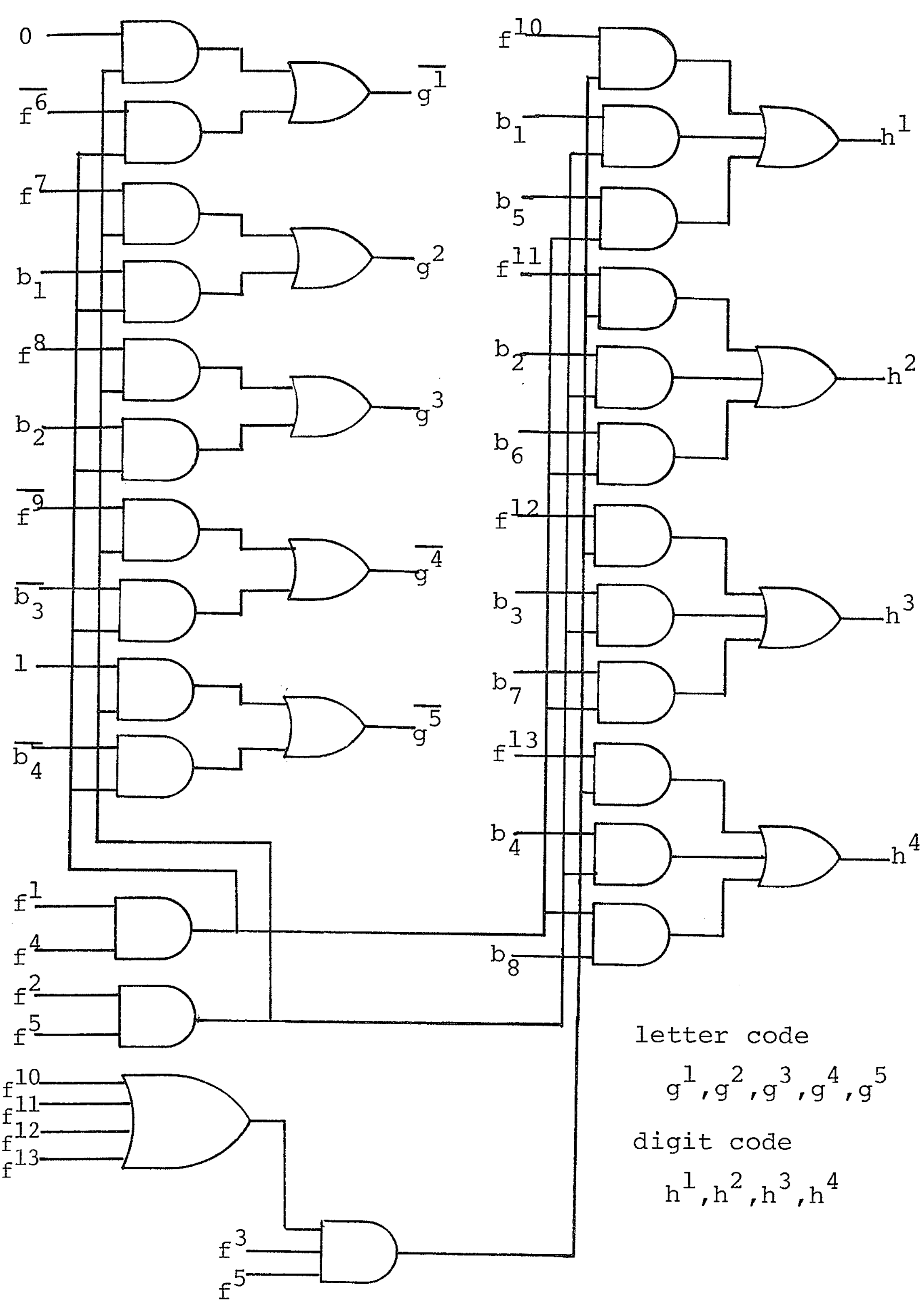


FIGURE 7.4

techniques has appeared. These algorithms produce circuits which compare quite well to circuits found by alternative techniques.

In the many-valued case, an algorithm for the synthesis of single-output circuits composed of universal decision elements has been presented. The importance of this algorithm is that it demonstrates two-place decomposition is a viable approach to the synthesis of many-valued combinational switching circuits. It forms a basis for future research into this problem. Extensions to other hardware schemes are currently of most interest to the author.

This work has shown there is merit in a unified approach to two-valued and many-valued logic design problems. Traditionally, these have been treated as distinct research areas. While certain implementation details will no doubt differ in the solutions to two-valued and many-valued problems, the underlying structure will often be the same. This was the case for the algorithms presented in this thesis. We found that a unified approach resulted in a better understanding of the synthesis problem, and greatly aided in developing a solution.

REFERENCES

[1] C.M. Allen, and D.D. Givone, "A minimization technique for multiple-output logic systems," IEEE Trans. Elec. Comp., V. EC-17, pp. 182-184, 1968.

[2] R.L. Ashenhurst, "The decomposition of switching functions," Annals. Harvard Comp. Lab., V. 29, pp. 74-116, 1959.

[3] D.F. Barnard, and D.F. Holman, "The use of Roth's decomposition algorithm in multi-level design of circuits," Comp. J., V. 11, pp. 269-276, 1968.

[4] T.C. Bartee, "Computer design of multiple output logic networks," IRE Trans. Elec. Comp., V. EC-10, pp. 21-30, 1961.

[5] A.K. Choudhury, and S.R. Das, "Determination of one of the minimal solutions of switching functions," Int. J. Control, V. 1, pp. 556-583, 1965.

[6] N. Christofides, "An algorithm for the chromatic number of a graph," Comp. J., V. 14, pp. 38-39, 1971.

[7] A. Church, "Conditional disjunction as a primitive connective for the propositional calculus," Portugaliae Math., V. 7, pp. 87-90, 1948.

[8] D.G. Corneil, and B. Graham, "An algorithm for determining the chromatic number of a graph," SIAM J. Comp., V. 2, pp. 311-318, 1973.

[9] H.A. Curtis, A New Approach to the Design of Switching Circuits, Van Nostrand, Princeton N.J., 1962.

[10] H.A. Curtis, "A generalized tree circuit," JACM, V. 8, pp. 484-496, 1961.

[11] E.S. Davidson, "An algorithm for NAND decomposition under network constraints," IEEE Trans. Elec. Comp., V. EC-18, pp. 1098-1109, 1969.

[12] D.L. Dietmeyer, and P.R. Schneider, "A computer oriented factoring algorithm for NOR logic design," IEEE Trans. Elec. Comp., V. EC-14, pp. 868-874, 1965.

[13] D.L. Dietmeyer, and Y.H. Su, "Logic design automation of fan-in limited NAND networks," IEEE Trans. Elec. Comp., V. EC-18, pp. 11-22, 1969.

[14] C.R. Edwards, "The design of logic networks based upon Rademacher-Walsh spectral techniques," Internal Report, School of Elec. Eng., U. of Bath, England, 1974.

[15] D. Etiemble, and M. Israel, "Implementation of a complete ternary algebra with elementary operators - application to ternary flip-flop," Proc. 1975 Int. Symp. Multiple-Valued Logic, Bloomington Indiana, pp. 316-329.

[16] R. Fridshal, "The Quine algorithm," Summaries of Talks at the Summer Inst. of Formal Logic, Cornell U., pp. 211-212, 1957.

[17] G. Frieder, A. Fong, and C.Y. Chao, "A balanced ternary computer," Proc. 1973 Int. Symp. Multiple-Valued Logic, Toronto Ont., pp. 68-88.

[18] M.J. Ghazala, "Irredundant disjunctive and conjunctive

forms of a Boolean function," IBM Res. and Dev., V. 1, pp. 171-176, 1957.

[]9] I. Halpern, and M. Yoeli, "Ternary arithmetic unit," Proc. IEE, V. 115, pp. 1385-1388, 1968.

[20] F. Harary, Graph Theory, Addison-Wesley, Reading Mass., 1972.

[21] M. Harrison, Introduction to Switching and Automata Theory, McGraw Hill, New York N.Y., 1965.

[22] S.L. Hight, "Complex disjunctive decomposition of incompletely specified Boolean functions," IEEE Trans. Elec. Comp., V. EC-22, pp. 103-110, 1973.

[23] IBM, "FORTRAN G & H programmers guide," System Support Document GC28-6817-3.

[24] IBM, "IBM system/360 and system/370 FORTRAN IV language," System Support Document GC28-6515-10.

[25] IBM, "IBM system/370 principles of operation," System Support Document GA22-7000-3.

[26] IBM, "OS/VS - DOS/VS - VM/370 assembler language," System Support Document GC33-4010-2.

[27] S.L. Hurst, Private Communication.

[28] M. Karnaugh, "The map method for the synthesis of combinational logic circuits," Trans. AIEE, V. 72, pp. 593-598, 1953.

[29] R.M. Karp, "Functional decomposition and switching circuit design," SIAM J., V. 11, pp. 291-335, 1963.

[30] 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. Switching Circuit Theory and Logic Design, pp. 152-162, 1961.

[31] J. Loader, "Second order and higher order universal decision elements in m-valued logic," Proc. 1975 Symp. Multiple-Valued Logic, Bloomington Indiana, pp. 53-57.

[32] J. Loader, "Universal decision elements in m-valued logic," Zeitschr.f.math.logik und Grundlagen d.Math., V. 20, pp. 1-18, 1974.

[33] F. Luccio, "A method for the selection of prime implicants," IEEE Trans. Elec. Comp., V. EC-18, 1966.

[34] D.W. Matula, G. Marble, and J.D. Isaacson, "Graph colouring algorithms," in R.C. Read(ed.) Graph Theory and Computing, Academic Press, New york N.Y., 1972.

[35] E.J. McCluskey, "Minimization of Boolean functions," Bell System Tech. J., V. 35, pp. 1417-1444, 1956.

[36] D.M. Miller, "Decomposition techniques for NAND circuits," M.Sc. Thesis, Dept. Comp. Sci., U. of Manitoba, 1973.

[37] D.M. Miller, "An algorithm for determining the chromatic number of a graph," Technical Report, School of Comp. Sci., U of New Brunswick, 1975 (also presented at the Fifth Manitoba Conf. on Numerical Math. and Comp., U of Manitoba, 1975).

[38] D.M. Miller, and J.C. Muzio, "Compatibility techniques

for the decomposition of ternary switching functions," Scientific Report No. 71, Dept. of Comp. Sci., U. of Manitoba, 1973.

[39] D.M. Miller, and J.C. Muzio, "Two-place decomposition of binary functions," Proc. Third Manitoba Conf. on Numerical Math., Winnipeg Man., 1973.

[40] D.M. Miller, and J.C. Muzio, "A fast method for determining the two-place decompositions of a binary function," Proc. Fourth Manitoba Conf. on Numerical Math. and Comp., Winnipeg Man., 1974.

[41] R.E. Miller, "State reduction for sequential machines," IBM Report RC-121, 1959.

[42] P. Morreale, "Partitioned list algorithms for prime implicant determination from canonical forms," IEEE Trans. Elec. Comp., V. EC-16, pp. 611-620, 1967.

[43] H.T. Mouftah, and I.B. Jordan, "Integrated circuits for ternary logic," Proc. 1974 Int. Symp. Multiple-Valued Logic, Morgantown W. Va., pp. 285-302.

[44] H.T. Mouftah, and I.B. Jordan, "A design technique for an integrable ternary arithmetic unit," Proc. 1975 Int. Symp. Multiple-Valued Logic, pp. 359-372.

[45] J.C. Muzio, and D.M. Miller, "Decomposition of ternary switching functions," Proc. 1973 Int. Symp. Multiple-Valued Logic, Toronto Ont., pp. 156-165.

[46] J.C. Muzio, and D.M. Miller, "A ternary universal decision element," Notre Dame J. Formal Logic, (forthcoming).

[47] J.C. Muzio and D.M. Miller, "Unary three-valued generators," (in preparation).

[48] N. Necula, "A numerical procedure for the determination of the prime implicants of a Boolean function," <u>IEEE Trans. Comp.</u>, V. EC-16, pp. 687-689, 1967.

[49] R.S. Nutter, and R.E. Swartwout, "A ternary logic minimization technique," <u>Conf. Rec. 1971 Symp. Theory and Applications of Multiple-Valued Logic</u>, Buffalo N.Y., pp. 112-123.

[50] D.L. Ostapko, R.G. Cain, and S.J. Hong, "A practical approach to two-level minimization of multivalued logic," <u>Proc. 1974 Int. Symp. Multiple-Valued Logic</u>, pp. 168-182.

[51] S.R. Petrick, "A direct determination of the irredundant forms of a Boolean function from the set of prime implicants," <u>Tech. Rep. No. 56-110</u>, AF Cambridge Research Centre, Bedford Mass., 1956.

[52] E.L. Post, "Introduction to a general theory of elementary propositions," <u>Amer. J. Math.</u>, V. 43, pp. 163-185, 1921.

[53] G. Povarov, "On the functional decomposition of Boolean functions," <u>Dokl. Akad. Nauk</u>, V. 94, pp. 801-803, 1954.

[54] I.B. Pyné, and E.J. McCluskey, "The reduction of redundancy in solving prime implicant tables," <u>IRE Trans. Elec. Comp.</u>, V. EC-11, pp. 473-482, 1962.

[55] W.V. Quine, "The problem of simplifying truth functions," Am. Math. Monthly, V. 59, pp. 521-531,1952.

[56] W.V. Quine, "A way to simplify truth functions," Am. Math. Monthly, V. 62, pp. 627-631, 1955.

[57] A. Rose, "Sur les éléments universal trivalent de décision," Comptes Rendus, V. 269, pp. 1-3, 1969.

[58] J.P. Roth, "Algebraic topological methods for the synthesis of switching systems I," Trans. Am. Math. Soc., V. 88, pp. 301-326, 1959.

[59] J.P. Roth, and R.M. Karp, "Minimization over Boolean graphs," IBM Res. & Dev., V. 6, pp. 227-238, 1962.

[60] 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., V. 4, pp. 326-344, 1959.

[61] F. Scheid, Theory and Problems of Numerical Analysis, McGraw-Hill, New York N.Y., 1968.

[62] P. Sebastian, and Z.G. Vranesic, "Ternary logic in arithmetic units," Proc. 1972 Symp. Theory and Applications of Multiple-Valued Logic," Buffalo N.Y., pp. 153-159.

[63] W. Semon, "Characteristic numbers and their use in the decomposition of switching functions," Proc. ACM, pp. 273-280, 1952.

[64] C.E. Shannon, "A symbolic analysis of relay and switch-

ing circuits," Trans. AIEE, V. 57, pp. 713-723, 1938.

[65] C.E. Shannon, "The synthesis of two-terminal switching circuits," Bell System Tech. J., V. 28, pp. 59-98, 1949.

[66] T. Singer, "The decomposition chart as a theoretical aid," Rep. No. BL-4, Harvard Comp. Lab., pp. 1-28, 1953.

[67] T. Singer, "Some uses of truth tables," Ann. Comp. Lab. Harvard U., V. 29, p. 30, 1959.

[68] W.R. Smith III, "Minimization of multivalued functions," Proc. 1974 Int. Symp. Multiple-valued Logic, pp. 27-44.

[69] B. Sobociński, "On a universal decision element," J. of Comp. Systems, V. 1, pp. 71-80, 1953.

[70] S.Y.H. Su, and P.T. Cheung, "Computer-oriented algorithms for minimizing multiple-valued switching functions," Conf. Rec. 1971 Symp. Theory and Applications of Multiple-valued Logic, Buffalo N.Y., pp.140-152.

[71] S.Y.H. Su, and P.T. Cheung, "Computer minimization of multi-valued switching functions," IEEE Trans. Elec. Comp., V. EC-21, pp. 995-1003, 1972.

[72] Y.H. Su, and D.L. Dietmeyer, "Computer-oriented algorithms for synthesizing multiple-output switching circuits," IEEE Trans. Elec. Comp., V. EC-18, pp. 58-63, 1969.

[73] S.Y.H. Su, and C. Nam, "Computer-aided synthesis of multiple-output multilevel NAND networks with fan-in

and fan-out constraints," IEEE Trans. Elec. Comp., V. EC-20, pp. 1445-1455, 1971.

[74] S.Y.H. Su, and R.A. Weingarten, "Design of multiple-output logic networks by a computer with a small memory space," Proc. IEEE Region IV Conf., Sacremento Calif., 1971.

[75] Texas Instruments Incorporated, The TTL Data Book for Design Engineers, Texas Instruments Incorporated, Dallas Texas, 1973.

[76] G.C. Vandling, "The simplification of multiple-output switching circuits composed of unilateral devices," IRE Trans. Elec. Comp., V. EC-3, pp. 6-12, 1954.

[77] V.I. Varshavskiy, "Functional divisibility in three-valued logic," Izvestija A.N.N., S.S.S.R., Techn. Kibernetika, V. 2, pp. 39-42, 1965.

[78] E.W. Vietch, "A chart method for simplifying truth functions," Proc. ACM, pp. 127-133, 1952.

[79] Z.G. Vranesic, and V.C. Hademacher, "Ternary logic in parallel multipliers," Comp. J., V. 15, pp. 254-258, 1972.

[80] Z.G. Vranesic, and K.C. Smith, "Engineering aspects of multi-valued logic systems," Computer, V. 7, No. 9, pp. 34-41, 1974.

[81] C.C. Wang, "An algorithm for the chromatic number of a graph," JACM, V. 21, pp. 385-391, 1974.

[82] K.M. Waliuzzaman, and Z.G. Vranesic, "On decomposition of multi-valued switching functions," Comp. J., V. 13, pp. 359-362, 1970.

[83] D. Zissos, and F.G. Duncan, "Boolean minimization," Comp. J., V. 16, pp. 174-179, 1973.

[84] T.M. Whitney, "Introduction to calculators," appearing in Introduction to Computer Architecture (H.S. Stone editor), Science Research Associates, Chicago Illinois, 1975.

[85] A. Mukhopadhyay, "Symmetric ternary switching functions," IEEE Trans. Elec. Comp., V. EC-15, pp.731-739, 1966.